FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Data Mining techniques for Prediction of secondary and 3D structures of proteins

**Luís Miguel da Costa Oliveira**




Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Camacho

June 27, 2018

# Data Mining techniques for Prediction of secondary and 3D structures of proteins

**Luís Miguel da Costa Oliveira**

Mestrado Integrado em Engenharia Informática e Computação

June 27, 2018

# Abstract

Proteins are large biomolecules that intervene in most of the different functions within organisms. They consist of one or more long chains of amino acids. The function of a protein in a living organism is linked with its three-dimensional (3D) structure.

Protein structure prediction is the inference of a protein's 3D structure from its linear sequence of amino acids. Before achieving the 3D shape, several secondary structures are created: helices and sheets. This is seen as one of the main goals of molecular biology and, as such, the scientific community has been investing in the development of algorithms that are capable of making this prediction. Knowing the rules (if they actually exist) of protein folding would play a big part in enabling to synthesize medicines for all of the major diseases known today.

This project aims to tackle the problem through the application of data mining classification algorithms using data retrieved from available databases in the Web. The goal is to contribute to this scientific area by developing one more methodology to predict protein structures as accurately as possible. Data mining serves the purpose of finding patterns in large data sets with the intent to transform that data into understandable information. In the context of this specific area of study, data mining assumes a huge importance since the factors that determine the correlation between the linear sequence of amino acids and the corresponding three-dimensional structure of the protein have yet to be discovered therefore making the discovery of patterns in the data sets so significant.

The results found by this work proved fruitful since it was possible to create classification models for protein structure prediction with a good accuracy level and in relatively small amounts of time.

# Resumo

As proteínas são biomoléculas de grande dimensão que intervêm na grande maioria das funções de um organismo. Elas são constituídas por uma ou mais grandes cadeias de aminoácidos. A função desempenhada pela proteína num ser vivo está ligada à sua estrutura tridimensional (3D).

A previsão da estrutura de uma proteína é a inferência da estrutura 3D de uma proteína a partir da sua sequência linear de aminoácidos. Antes de atingir a forma 3D, várias estruturas secundárias são criadas: hélices e folhas. Esta área é vista como um dos principais objectivos da biologia molecular e, nesse sentido, a comunidade científica tem investido no desenvolvimento de algoritmos capazes de realizar esta previsão. Saber as regras (se é que existem) da dobragem das proteínas teria um importante papel em permitir sintetizar medicamentos para muitas das doenças conhecidas na actualidade.

Este projecto visa atacar este problema através da aplicação de algoritmos de *data mining* para classificação usando dados recolhidos de repositórios disponíveis na *web*. O objectivo é contribuir para esta área científica desenvolvendo mais uma metodologia para a previsão de estruturas de proteínas com a maior exactidão possível. *Data mining* serve o propósito de encontrar padrões em grandes conjuntos de dados com a intenção melhorar o seu entendimento. No contexto desta área de estudo específica, *data mining* assume uma grande importância uma vez que os factores que determinam a correlação entre a sequência linear de aminoácidos e a estrutura tridimensional correspondente da proteína ainda estão por descobrir, tornando a descoberta de padrões tão significativa.

Os resultados deste trabalho provaram-se frutíferos, uma vez que foi possível criar modelos de classificação para a previsão de estruturas das proteínas com bons níveis de exactidão e em relativamente pouco tempo.

# Acknowledgements

I would like to thank to FEUP for the opportunities it gives to its students to learn, grow and meet all kinds of people that contribute to form young adults with values. I would also like to thank to the city of Porto which I now feel like it is a second home.

I leave a word of appreciation to Professor Rui Camacho for all the help and guidance provided during this project.

To my friends, with whom I lived beautiful moments during five spectacular years, I thank you for everything we went through together and all the help in times of need. Much more is yet to come. Exciting times are ahead for us.

Last but definitely not least, I thank my parents for all the sacrifices made, all the unconditional support and for always making me feel loved. This work is a little bit theirs too. Filipe "Loiro" and Ni, I love you from the bottom of my heart.

Luís Oliveira

*"Eles não sabem que o sonho*
*é uma constante da vida"*

António Gedeão

# Contents

# List of Figures

LIST OF FIGURES

# List of Tables

# Abbreviations

| | |
|---|---|
| API | Application programming interface |
| ARFF | Attribute-Relation File Format |
| ASCII | American Standard Code for Information Interchange |
| AUC | Area under the curve |
| AUROC | Area Under the Receiver Operating Characteristic Curve |
| CRISP-DM | Cross-industry standard process for data mining |
| CSV | Comma-separated values |
| JSON | JavaScript Object Notation |
| KDD | Knowledge discovery in databases |
| *k*-NN | *k*-nearest neighbors algorithm |
| NMR | Nuclear magnetic resonance |
| PDB | Protein Data Bank |
| RMSD | Root mean square deviation |
| SD | Standard deviation |
| SVM | Support Vector Machine |
| URL | Uniform Resource Locator |

# Chapter 1

# Introduction

This dissertation is inserted in the area of Bioinformatics. This section provides a description of its relevance and the context of the scientific area, it is explained what is intended to do and what should be accomplished as well as a general description of the structure of the dissertation.

## 1.1 Context

Bioinformatics is a field of studies that combines the knowledge of several other areas of science like computer science, biology and engineering with the intent to create methods and tools to help in the understanding of biological data. The theme of this dissertation falls in this scientific field taking into account that it is expected to capitalize on the advantages provided by data mining algorithms to improve the prediction of protein structures.

Proteins are big biomolecules that exist in living organisms and they assume a great variety of functions in cellular biology. These functions are dependent of each protein's 3D shape. Protein folding is the name given to the set of events that turn a single or group of linear sequences of amino acids into a three dimensional structure. Flaws in this process may, for example, result in a wide array of different diseases.

Protein structure prediction is an issue that has been attracting attention of the scientific community for many years. It is regarded as one of the biggest unsolved problems of modern science [Edi05]. Regarding the prediction, various techniques already exist and this is the subject that will be approached in this dissertation.

## 1.2 Motivation

Protein structure prediction is seen today as one of the greatest challenges in Molecular Biology. The determination of the 3D structure of a protein is a vital element for various aspects of current research in the biology field. Though millions of protein sequences exist in databases just a few

thousands have structures that were experimentally determined. The pressing need of biologists for 3D models of proteins has brought extra significance to computer-based methods of predicting protein structures.

One of the primary motivations for the development of this field is to replace time-consuming costly biology experiments with quick low-cost computer simulations in hopes that the design of new drugs is accelerated and becomes more efficient.

Other examples of areas that could greatly benefit from the advances in this matter and demonstrate its biological usefulness include protein family assignment, drug screening, ligand docking and even in understanding biological mechanisms that are inherent to the appearance of multiple diseases.

## 1.3 Goals

The central goal of this work is to contribute to the scientific area with one more methodology for the prediction of protein structures.

Computer-based methods of protein structure prediction are processes that are normally slow and require a great amount of processing power. This means that there is great interest in constantly trying to enhance them. Progress towards automation and refinement of these techniques should prove valuable and definitely broaden the extent of modelable proteins. Taking this in consideration, another one of the goals of this work is to accelerate the computer-based prediction methods of the protein folding. Through the application of Data Mining algorithms to the data retrieved from available repositories it is expected to reduce the time required for this type of prediction.

## 1.4 Structure

Besides the introduction where context is given and the motivation and goals of the dissertation are explained there are four other chapters.

In the second chapter is where light is shed upon key concepts for understanding the matter at hand as well as the state of the art of the scientific field.

The third chapter is where the proposed solution and its specification are described.

In the fourth chapter is where the results from the experiments are presented and analysed.

Finally, the fifth chapter is where the conclusions are taken and where the objectives satisfaction is evaluated. It is also where we leave some guidelines for possible future work.

# Chapter 2

# Protein structure prediction and Data Mining

In this chapter, basic necessary knowledge about the relevant areas of study for this work is provided and a description is made of the state of the art regarding the existing methods of determination of protein structures, prediction techniques, data mining tasks, adopted algorithms and frequently used tools.

## 2.1 Proteins

### 2.1.1 What are proteins?

Proteins are biomolecules of considerable size also known as macromolecules that are comprised of one or more sequences of amino acids (represented in Figure 2.1). These complex molecules that exist in every living organism execute a vast set of actions taking part in many of the chemical reactions at cellular level. They can serve various purposes assuming a catalyzing role speeding up or suppressing metabolic reactions, replicate DNA, adopt a role of transportation of other molecules and even play a part in differentiation and growth of cells.

There are numerous types of amino acids in proteins each and every one with singular chemical characteristics. These amino acids form long chains are linked to its neighbors by means of a covalent peptide bond. Proteins can, accordingly, be known as polypeptides. Every particular amino acids sequence gives way to a different protein and there are quite a few thousands.

Polypeptide backbone is the definition of a recurring series of atoms across the core of the polypeptide chain. Connected to this repeating sequence are the fractions of the amino acids that don't take part in making the bond and that confer to an amino acid its particular properties - the side chains. Both edges of a polypeptide chain differ in its chemical composition: the one that carries the free amino group ($NH_3^+$ that can also be written as $NH_2$ is called amino terminus or N-terminus, and the one carrying the free carboxyl group ($COO^-$ also referred as COOH) is the

carboxyl terminus or C-terminus. The sequence of amino acids associated with any given protein is at all times depicted from left to right, that is, N-to-C direction [AJL+08].

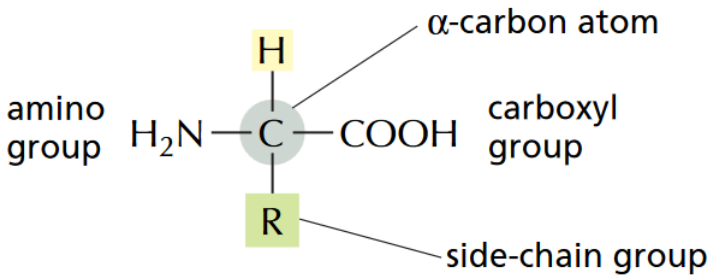


Figure 2.1: General formula of an amino acid [AJL+08]

### 2.1.2 Protein structure

Resulting from the interactions of the amino acids, the majority of proteins has a three-dimensional structure that is dictated by the organization of the amino acids sequence. The conformation, that is, the eventual folded arrangement of a given polypeptide chain is the state where the protein has typically reached its minimized free energy as it was coined by Anfinsen's "thermodynamic hypothesis" [Anf73].

Experiences with purified proteins regarding the protein folding process were run by biologists. Treating those proteins with some solvents that break up the non covalent bonds that hold the chain together in a specific conformation, makes those proteins unfold or denature. From this experiment a flexible polypeptide is what remains after losing its natural configuration. What happens when the denaturing agent is removed is that the polypeptide will frequently fold again, or renature, into the previous conformation. The process is depicted in Figure 2.2. This comes to prove that the amino acid chain encompasses all the necessary information for the specification of the three-dimensional shape of a protein.

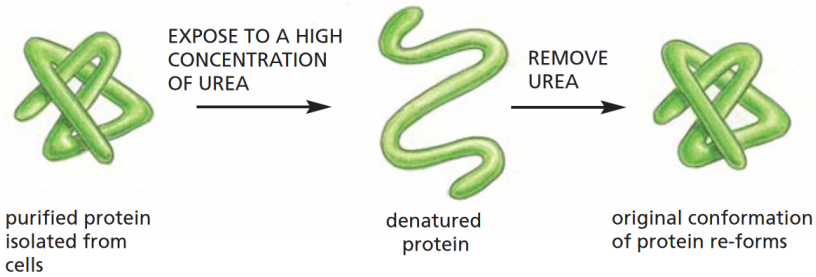


Figure 2.2: Refolding of a denatured protein [AJL+08]

Proteins come in a wide variety of shapes and its chains of amino acids and commonly are made up of 50 to 2000 single amino acids and, as such, they have amazingly complex folded configurations even for relatively small examples. Though this this could be thought of as a hurdle, there are ways of simplifying the representation of these structures. This happens since they are composed by combinations of various typical structures that are described next.

Upon comparing the 3D structure of plenty of distinct proteins, it is evident that, albeit the global conformation of each protein is exclusive, there are two frequently found patterns of fold found in some of its parts. These two local segments represent the secondary structure of a protein. The first pattern to be found was the $\alpha$ helix. Around one year later another recurring pattern was discovered and named $\beta$ sheet. Both of the patterns are very common on the grounds that they are a product of hydrogen-bonding between amino hydrogen and carboxyl oxygen atoms in the polypeptide backbone and don't involve the amino acids' side chains.

- $\alpha$ helix (depicted in Figure 2.3)

  This is the most prevalent local structure and is also the easiest to predict from the linear sequence of amino acids. An $\alpha$ helix is created when a polypeptide curls on itself and arranges itself as a hard cylinder. At every four peptide bonds, a hydrogen bond occurs connecting the C=O of one bond to the N−H of the other bond. This makes way for a regular helix that has an amino acid every $100^{o}$, meaning that it completes a turn every 3.6 amino acids. These structures appear in large number in proteins like receptors or in charge of transportation close to cell membranes. The three dimensional structure of the protein characterized by its atomic coordinates. These can be associated with the whole tertiary structure or just a protein domain.

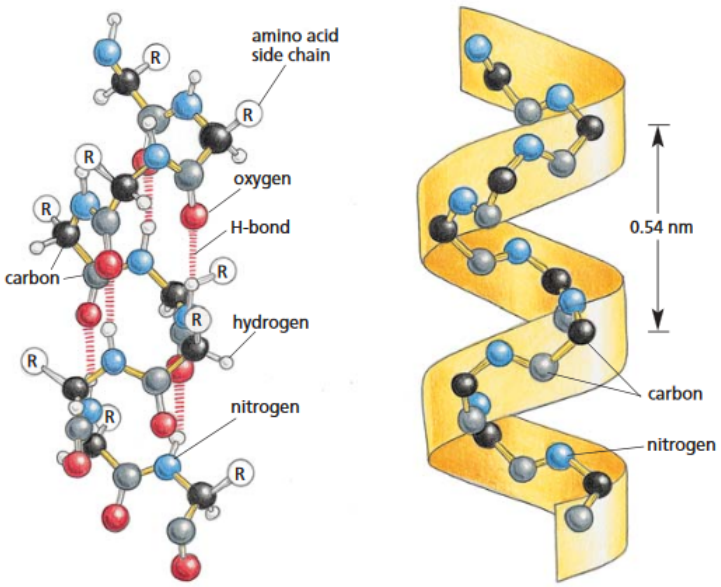


Figure 2.3: Representations of an $\alpha$ helix [AJL+08]

- β sheet (depicted in Figure 2.4)

  This structure is composed of nearly completely extended β strands. The strands can relate in a parallel or antiparallel orientation and each one has a associated pattern of hydrogen bonding. Antiparallel sheets have hydrogen bonds perpendicular to the strands, and bond pairs alternate between narrowly spaced and widely spaced. Parallel β sheets have equally spaced hydrogen bonds [Ric81].

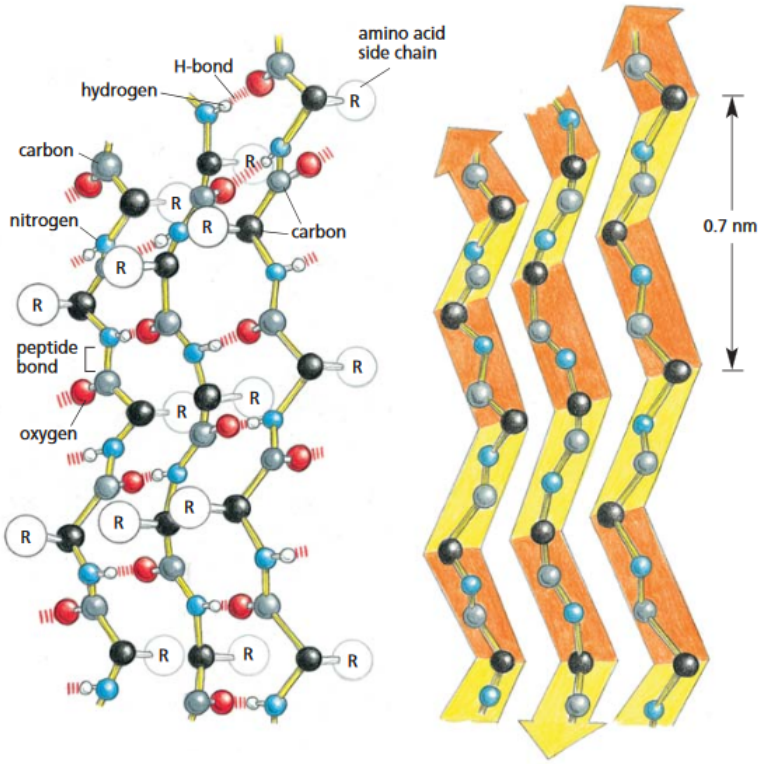


Figure 2.4: Representations of a β sheet [AJL+08]

The three dimensional shape of a protein is also known as its tertiary structure. In this state, the protein shall have one single polypeptide backbone that has one or more of the secondary structures mentioned before. The amino acid side chains of the backbone can bond and interact in a huge amount of ways. The way those come to take place is, in effect, related with the function the protein assumes.

### 2.1.3 Protein structure prediction

Protein structure prediction is the inference of a protein three dimensional structures from its amino acids linear sequence that, as mentioned before, contains all the necessary information for it. The problem, though, lies in how to achieve this. Protein structure prediction is one of the major goals of Molecular Biology nowadays and it has been that way for many years.

The difficulty that is posed in this field of science is well described in Levinthal's paradox that states that because of the great number of degrees of freedom that an unfolded polypeptide

chain has, each molecule can have an enormous amount of possible conformations [Lev69]. He estimated that a protein with 150 amino acids could have about $10^{300}$ different conformations and that it would take a colossal amount of time to process which of the possibilities would be the correct conformation. What makes it a paradox is that the folding of a protein occurs spontaneously in a small fraction of a second.

In 1973, Anfinsen had already stated that the "ultimate aim of the enzymologist and the protein chemist is to be able to synthesize an amino acid sequence that, when allowed to fold, will assume a stable predesigned three dimensional arrangement of atoms capable of carrying out the desired catalytic act" [Anf73] and this still holds true to this day. In fact, evolution in protein structure prediction can prove itself useful in a lot of areas like biomedicine, be it protein family assignment, meticulous drug design or even in understanding the cellular level events that lead to the appearing of many diseases due to errors occurred in the protein folding process [CFR+12].

### 2.1.4 3D protein structure determination techniques

Out of the known protein 3D structures available in repositories today, the great majority was determined through an experimental method called X-ray crystallography [PDBa]. Taking into consideration that a lot of materials can produce crystals, likewise many organic, inorganic and biological molecules, X-ray crystallography has played a big role in the advancement of various scientific fields.

The objective of this technique is to obtain the three dimensional structure out of a crystal. X-ray beams are used on purified samples at a elevated concentration that are crystallized. From the places where the diffraction occurs patterns can be recognized. By measuring the intensity and angles of the beams diffracted, a skilled crystallographer can, based on the crystal's electron density, create a three dimensional picture. Based on this, the mean positioning of the atoms can be resolved along with some other informations like their chemical bonds. The precision of the aforementioned picture can be bettered using some methods up to the point that it becomes clear enough to allow the construction of the 3D structure of the amino acids sequence. This conformation can then be improved to better suit the picture and to assume a favorable thermodynamic conformation [SM00].

### 2.1.5 Protein structure prediction techniques

Protein structure prediction techniques can vary. These computational approaches that sometimes even intertwine themselves oscillate between two types with are described in the following points.

#### 2.1.5.1 *Ab initio* (or *de novo*) prediction

*Ab initio* literally means "from the beginning". This approach tries to predict the three dimensional structure of a protein only from the its linear sequence of amino acids. In this method, the community tries to achieve the results by using only knowledge that is supported by physics and does not use information available in databases. The results obtained from theses processes are

still limited since they require a lot of computational power and there is still some inconsistency with the force-fields. Even with these obstacles there have been successful cases that are noteworthy. Achievements have too existed in methods that mix the potentialities of the physical approach with data from repositories. Even considering that these physical experiments are trailing behind other approaches of the bioinformatics area, the functions that calculate the energy are revealing themselves as more convincing that they were initially thought of [Dil07].

#### 2.1.5.2 Homology modeling

On the other end of the spectrum are the homology modeling methods. These procedures try to construct a possible 3D structure for the protein by comparing the "target" protein to a "template" - template based modeling. This type of modeling depends on the recognition on know structure probable to be similar to the target one, and on the construction of an alignment which maps amino acids in the target sequence to the ones on the template. The quality of these predictions depends directly of the accuracy of the alignment and of the template chain. A typical measurement that is used is the root mean square deviation (RMSD).

In hopes of improving these comparison models and their RMSD, the resulting structures have been subject to molecular dynamics software that are used for simulations that compare them against the experimentally discovered conformations. It has been concluded that the information existent in the Protein Data Bank (PDB) is at least enough for solving the 3D structure of single-domain proteins. Working in bettering these computer modeling techniques can, therefore, pay great dividends and drastically improve the amount of modelable proteins [Zha08].

## 2.2 Data Mining

### 2.2.1 What is data mining?

With the fast-paced advances in technology and the exponential growth of the Internet, the amount of data made available is made bigger by the day. With all this information being stored, the necessity of finding easier ways to navigate it and to find useful knowledge turns greater too.

Following the idea described in the previous point that improving the modeling techniques can better the predicted results, data mining surges a good candidate to help in that difficult path.

Data mining can be described as "the discovery of interesting, unexpected or valuable structures in large datasets". State of the art data mining contrasts with common statistics since it aggregates that field with others from computer science [Han07]. This important area of studies is also often referred to as knowledge discovery in databases (KDD). It grants the capacity of analyzing and visualize very big quantities of data without bias or with a predetermined hypothesis in mind.

### 2.2.2 Data mining tasks

Pattern finding is the number one objective of data mining. A pattern is a structure in a considerable amount of space that stands out from the rest because of its abnormal high occurrence and are usually "lost" in the middle of big quantities of data that has no interest. The decision of the pertinence of a given pattern always is dependent of the context. Different types of data call for different types of analysis.

Data mining can be thought of a way of completing six different kinds of tasks defined by Fayyad et al. in 1996. Those tasks are described in Table 2.1.

Table 2.1: The six tasks of data mining as defined by Fayyad et al. [FPSS96]

| | |
|---|---|
| Classification | Learning a function that maps (classifies) a data item into one of several predefined classes. |
| Regression | Learning a function that maps a data item to a real-valued prediction variable. The fit is as good as the correlation existent between the data. |
| Clustering | A descriptive task where one seeks to identify a finite set of categories or clusters to describe the data. |
| Summarization | Methods for finding a compact description for a subset of data, for example tabulating the mean and standard deviation for all fields |
| Dependency modeling | Finding a model that describes significant dependencies between variables. |
| Change and deviation detection | Discovering the most significant changes in the data from previously measured or normative values. |

### 2.2.3 CRISP-DM Methodology

CRISP-DM stands for CRoss-Industry Standard Process for Data Mining and is one of the leading data mining process models that people of this area use to tackle problems. It is a cycle that is divided into six different steps [CCK+00, AS08] (as depicted in Figure 2.5):

1. Business understanding – This initial phase focuses on understanding the project objectives and requirements from a business perspective, then converting this knowledge into a data mining problem definition and a preliminary plan designed to achieve the objectives.

2. Data understanding – The data understanding phase starts with an initial data collection and proceeds with activities in order to get familiar with the data, to identify data quality problems, to discover first insights into the data or to detect interesting subsets to form hypotheses for hidden information.

3. Data preparation – The data preparation phase covers all activities to construct the final dataset from the initial raw data.

4. Modeling – In this phase, various modeling techniques are selected and applied and their parameters are calibrated to optimal values.

5. Evaluation – At this stage the model (or models) obtained are more thoroughly evaluated and the steps executed to construct the model are reviewed to be certain it properly achieves the business objectives.

6. Deployment – Creation of the model is generally not the end of the project. Even if the purpose of the model is to increase knowledge of the data, the knowledge gained will need to be organized and presented in a way that the customer can use it.

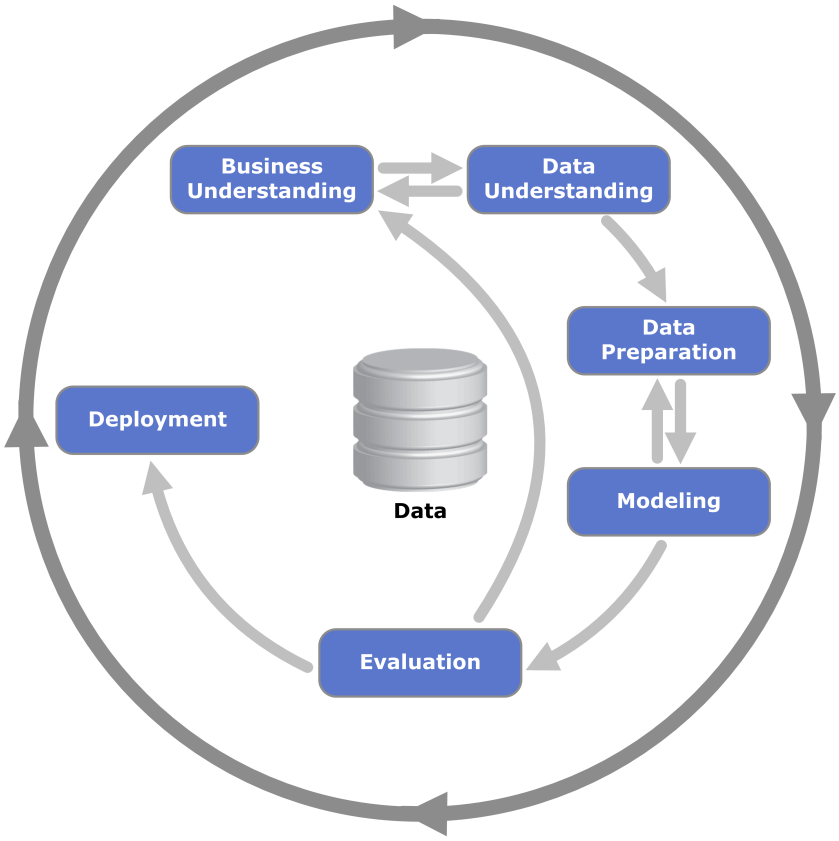


Figure 2.5: The CRISP-DM life cycle

### 2.2.4 Relational data mining

Data in its most traditional fashion is represented in a matrix form which means that rows describe observations and columns normally represent variables. This manner of expressing data has several advantages and it's the most common one in statistics.

Despite the aforementioned, informations about real situations are rarely of this structure. Actually, there are frequently a lot of distinct entities about which different sorts of data are known.

Relational data mining aim is to study and find methods for KDD when a database comprises information about various types of objects. This is, most of the time, the case when we are speaking of a database that has multiple tables (relations). There are two ways in which these relations can be defined:

- Extensionally, by means of a table
- Intensionally, as explicit logical rules. These are normally relationships that we can infer from other relationships and are associated with some kind of general knowledge about the domain of the data. This type of general knowledge is typically named background knowledge or domain knowledge [Dže09].

## 2.3 Web Repositories

### 2.3.1 Protein Data Bank

The Protein data bank is a database for the three dimensional structural data of large biomolecules like proteins or nucleic acids.

The project first started in 1971 as an effort from a relatively small group of young crystallographers and has now, as of 2018, more than 140000 entries (as shown in Figure 2.6). The accessibility of the data and the ever growing need to understand it means that the PDB's community has come a long way since that small group of crystallographers. Nowadays, PDB is a crucial resource for researchers be it in academia or industry. It is used by teachers and students ranging from middle school to graduate school.

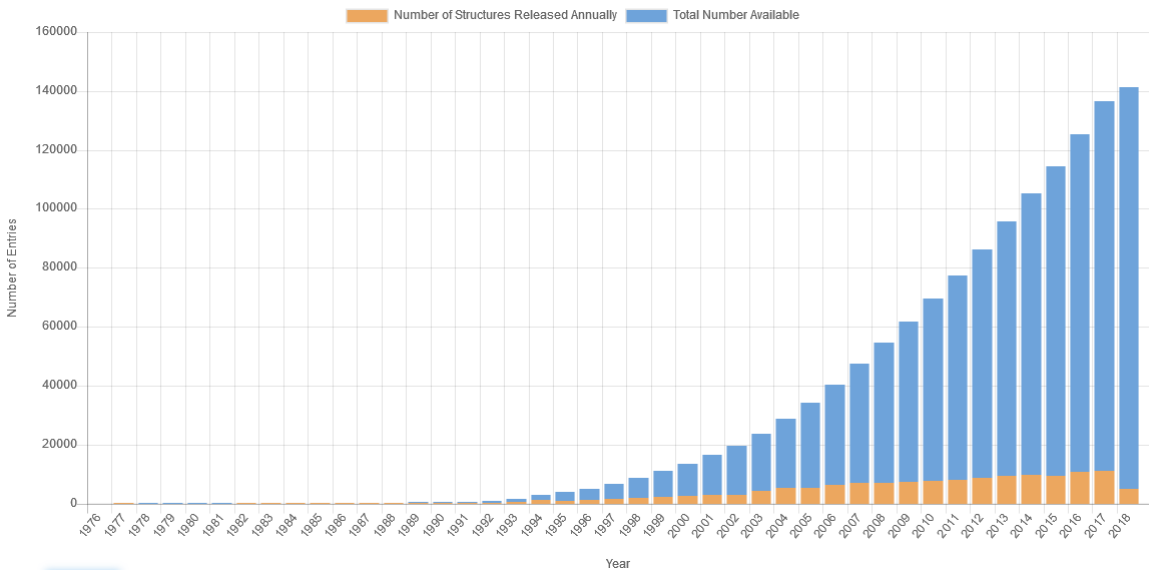


Figure 2.6: Overall Growth of Released Structures Per Year [PDBb]

This archive is a very good example of a shared resource that underwent through a big transformation over 40 years. Its evolution has been triggered by changes in science, technologies used

for the determination of the structures like nuclear magnetic resonance (NMR) methods, the type of structures that intend to be determined, different outlooks about data sharing and even the nature of the communities that show interest in this kind of data [Ber08].

### 2.3.2 Dunbrack Lab

The Dunbrack group concentrates on research in computational structural biology, including homology modeling, fold recognition, molecular dynamics simulations, statistical analysis of the PDB, and bioinformatics. In developing these methods, They use modern methods of Bayesian statistics and extensive benchmarking. They are interested in applying comparative modeling to important problems in various areas of biology, especially in cancer research [dun].

They make available in their website several precompiled lists of protein chains for various parameters culled from the PDB.

## 2.4 Tools for protein structure prediction

For the task at hand, and seeing that it is such a complex matter, several tools and softwares have been developed throughout the last decades. In this section will be list some that were found in a preliminary search for useful tools that could facilitate the job. They are divided into two different types: data mining tools and molecular dynamic ones.

### 2.4.1 Data mining tools

#### 2.4.1.1 RapidMiner

RapidMiner is a software launched in 2001 that provides ways of making data preparation, predictive analysis and a good visual interface that provides a nice analysis of results.

#### 2.4.1.2 WEKA

Waikato Environment for Knowledge Analysis (WEKA) is a free license suite that offers tools allowing to perform various data mining tasks. like clustering, regression, classification and visualization.

This software provides implementations of learning algorithms that can easily be applied to a dataset. It also makes available a wide set of tools for editing datasets, such as algorithms for discretization and sampling. It is possible to preprocess a dataset, feed it into a learning scheme, and analyze the resulting classifier and its performance without even writing a single line of code.

The WEKA workbench provides methods for the principal data mining problems: regression, classification, clustering, association rule mining and attribute selection. Since knowing the data is such an important part of the data mining process, WEKA supplies various data visualization options and data preprocessing tools. Every algorithm takes as input a single relational table that can be read from a file or originated by a database query [WFE05].

### 2.4.2 Molecular dynamic tools

#### 2.4.2.1 GROMACS

GROMACS which stands for Groningen Machine for Chemical Simulations is a molecular dynamics simulator that was developed with the intent to work with lipids, nucleic acids and proteins. It is distributed as free software and is one of the fastest available [BvdSvD95].

## 2.5 Classification Model Validation

Since the most relevant data mining task for the problem we aim to tackle is classification which was already succinctly described in the Table 2.1, in this section we will dive into more specific details about how classification models are validated.

### 2.5.1 Cross-Validation

This is one of the model validation techniques used to evaluate how the results of a classification algorithm will generalize when applied to a real world problem or, in other words, an independent dataset.

Since this method is usually utilized in contexts where one wants to figure out how accurately will perform a prediction, it is normally provided with two dataset to emulate that behavior:

- A dataset of known data and how it classifies on which the training is run (training set);
- A dataset of unknown data which we wish to classify and on which the model is tested (testing set).

In an attempt to reduce variability, most methods have various rounds of cross-validation working with different parts of the dataset and the results are an average of the several rounds so we can have an approximation of the predictive performance.

There are different ways the initial dataset can be divided and it depends on the method chosen. Taking into account that the amount of data we are going to work with is relatively large, bellow we will focus only on explaining how the non-exhaustive methods work. Being non-exhaustive means that they do not compute all the possible splits of the original dataset.

#### 2.5.1.1 Holdout method

In this method the initial dataset is randomly split into two subsets that are mutually exclusive. These two subsets, the training set and the test set, do not have to have the same dimension. In fact, it is very frequent to have a training set that is larger than the test set.

#### 2.5.1.2 *k*-fold method

In this method the original dataset is equally split into $k$ subsamples with the same size.

From the $k$ samples, one of them is taken as the test set opposed to the remaining $k$-1 samples which are used as the training set. This process happens $k$ times with each sample being used once as the test set. If a single estimation is desired, the $k$ results can be averaged.

#### 2.5.1.3 Monte Carlo method

In this method we repeatedly do a random subsampling of the original dataset splitting it into training and test data. With each of those splits, a model is trained with the trained set and applied to the test set. The final results are an average of the results gathered with each random subsample.

The advantage of this method over the $k$-fold method is that in this case the proportion of the training/test split is not dependent of the number of folds. On the other hand, in this method we cannot be sure that every observation is used for testing the model. Some might even be used more than once.

## 2.6 Classification Model Metrics

The performance of a model or classification algorithm can be assessed through a wide variety of metrics. To understand them fully, it is necessary to understand the concept of confusion matrix exemplified in Table 2.2.

Table 2.2: Confusion matrix example

| | | Actual Class | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted Class | Positive | True Positive (TP) | False Positive (FP) |
| | Negative | False Negative (FN) | True Negative (TN) |

In order to clarify beyond any doubt and for better understanding of the metrics that will be explained, what is shown in the confusion matrix is the following:

- TP is the number of positive instances correctly identified as positive
- FP is the number of negative instances wrongly identified as positive
- FN is the number of positive instances wrongly identified as negative
- TN is the number of negative instances correctly identified as negative

### 2.6.1 Accuracy

Accuracy is related with how close the predicted values are to the real ones.

When applied to classification, it demonstrates the fraction of correctly classified instances.

$$ACC = \frac{TP+TN}{TP+TN+FP+FN}$$

### 2.6.2 Precision

Precision is related with how close the predicted values are to each other. It is important to not mistake precision for accuracy seeing as they are independent metrics and, therefore, a dataset can be either accurate, precise, both or none.

In the context of classification, precision is the positive predictive value (PPV).

$$PPV = \frac{TP}{TP+FP}$$

### 2.6.3 True positive rate

True positive rate (TPR) measures the proportion of positives that are correctly classified as such. In other words, the higher TPR, the fewer positive data instances will be missed.

$$TPR = \frac{TP}{TP+FN}$$

### 2.6.4 False positive rate

False positive rate (FPR) measures the proportion of negatives that are wrongly classified as positives. In other words, the higher FPR, the more negative data instances will be misclassified.

$$FPR = \frac{FP}{FP+TN}$$

### 2.6.5 F-measure

F-measure is the harmonic mean of the precision and the true positive rate.

$$F-measure = 2 \times \frac{PPV \times TPR}{PPV+TPR}$$

### 2.6.6 Area Under the Receiver Operating Characteristic Curve

This metric relates the true positive rate with the false positive rate in a way that the area under the curve (AUC) (depicted in blue as an example in Figure 2.7) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.

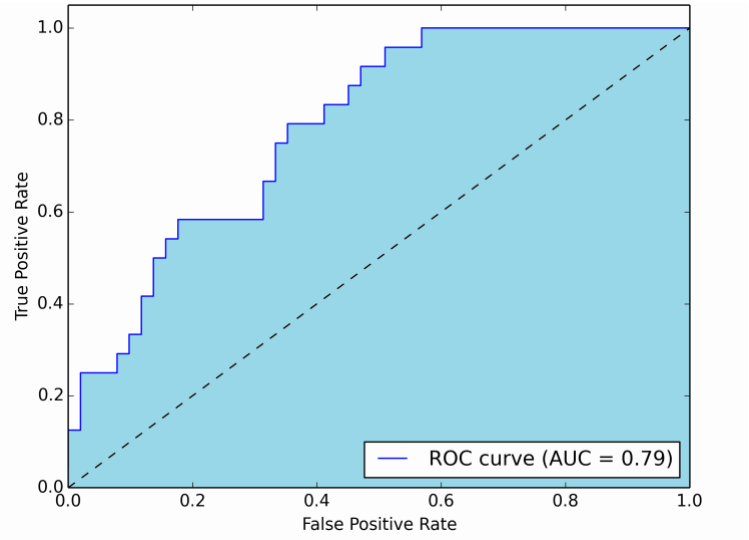


Figure 2.7: Receiver operating characteristic example

## 2.7 Classification algorithms

In this section, data mining classification algorithms that are considered relevant to the problem will be described briefly with the intent of providing some background understanding of what they do and how they perform it.

### 2.7.1 C4.5

C4.5 is an extension of an earlier algorithm called ID3, both of which were developed by Ross Quinlan [Qui92].

This algorithm generates a decision tree that can be run for classification. It is probably the most used algorithm in practical cases today [WFE05].

The decision trees this algorithm creates have its origin in a training dataset. For every node of the tree, C4.5 picks the attribute of the data in question that better partitions the group of samples. This attribute is chosen with regard to the normalized information gain. The one with the highest value is the attribute that is used. Afterwards the algorithm proceeds to do the same for each of the new subsets of data that resulted from the split.

### 2.7.2 Bagging

Bagging or bootstrap aggregating is an algorithm presented by Leo Breiman in 1994.

The algorithm was developed to help with the stability and accuracy of other algorithms.

Bagging generates new training sets of the same size by sampling the original dataset in a uniform manner and with replacement. This is known as a bootstrap sample. The model is then

built by combining the results from each bootstrap sample (either by averaging them or through voting)

Some of the advantages it provides to a various of widespread data mining algorithms like artificial neural networks and decision trees are an improvement in stability, accuracy and can even mitigate overfitting [Bre96].

### 2.7.3 Random forest

Random forests are an algorithm that perform by creating numerous decision trees and that makes use of bagging (described in Subsection 2.7.2) applied to those trees [Ho98]. Using that method of sampling helps better the results seeing as having many data subsets provides trees with a drastically minimized correlation.

An important aspect of the random forests and that actually differs from the ordinary bagging and makes the results interesting is that the algorithm picks a random set of features at each split in order to avoid that a feature is selected many times if it is a very strong predictor (which could cause a high correlation between the trees - precisely what is trying to be avoided) [Ho02].

### 2.7.4 Support vector machine

Support vector machine consists of supervised learning model that analyze and recognize patterns.

Given a starting dataset, for each record the algorithm predicts to which of two possible classes it belongs, thus making it a non-probabilistic binary linear classifier.

This kind of model represents the examples as points in space and finds a line of separation (called hyperplane) for the two classes. The gap created with this line is expected to be as wide as possible. This is exemplified in Figure 2.8 with the hyperplane in red.

New instances are classified by being mapped to that same space and depending on which side of the gap they fall, theirs class is predicted.

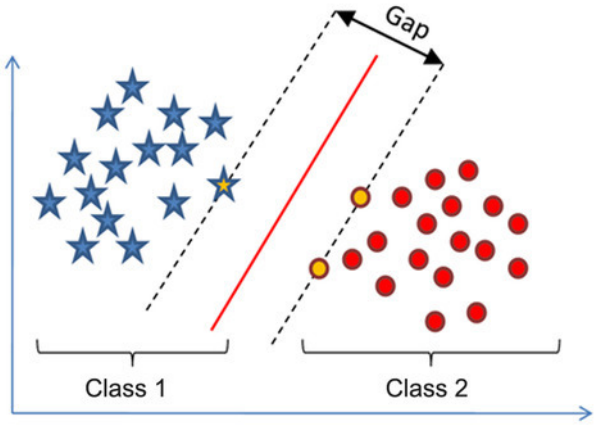


Figure 2.8: SVM example

### 2.7.5 *k*-nearest neighbors

The *k*-nearest neighbors algorithm (*k*-NN) in which the classification result is a class membership. The class to which an instance of the test set is classified into is decided by a majority vote taken by its *k* nearest neighbors. *k* is a positive integer value and usually it is a relatively small one [Alt92].

This algorithm is a kind of instance-based learning and is one of the simplest existing data mining methods.

A valuable addition that can be made to this algorithm is to establish different weights to the votes of the neighbors, making the nearest neighbors contribution greater than the that of the more distant ones.

A graphical example is provided in Figure 2.9 where the instance to be classified is depicted in green. With *k*=1 the new instance would be classified as Class 1 but with *k*=3 it would be classified as Class 2.

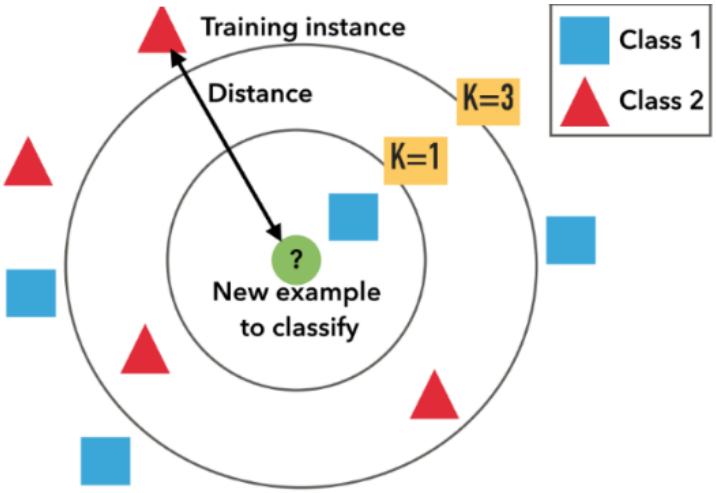


Figure 2.9: *k*-NN algorithm example

### 2.7.6 Artificial neural networks

Artificial neural networks (ANN) are a system that can learn how to run tasks when given examples. They are given this name because this system is somewhat inspired by the biological neural networks that exist in animal brains [Sch15].

These models are composed of processing unit (neurons) which are disposed in a direct graph structure (network) with weighted edges (synapses).

An ANN is used when the complete domain is not known in its entirety and because of that fact, the net has to learn with examples how to identify patterns. With each new instance, the ANN changes its weights so that it can minimize the errors in classifying.

### 2.7.7 AdaBoost

AdaBoost or Adaptive Boosting is an algorithm that was designed by Yoav Freund and Robert Schapire and that can be used together with a variety of other types of learning algorithms with the intent to improve the performance [SS99].

The results from other algorithms (weak learners) are combined in a weighted sum that equates to the final result of the boosted classifier. The adaptiveness of it is because every subsequent weak learner is improved in a way that enhances the classification of those instances that were misclassified by previous classification models.

## 2.8 File formats

### 2.8.1 PDB File

The PDB file (exemplified in Listing 2.1) made available by the PDB archive (already described in Subsection 2.3.1) was formulated in 1976 and is very intelligible, easy to read by humans and numerous computer applications use it. An entry in the PDB contains the *xyz* coordinate, information about the molecule's chemistry, its ligands, author remarks, etc.

```
HEADER    TRANSCRIPTION REGULATION                20-JAN-98   1A3C
TITLE     PYRR, THE BACILLUS SUBTILIS PYRIMIDINE BIOSYNTHETIC OPERON REPRESSOR,
TITLE    2 DIMERIC FORM
...
EXPDTA    X-RAY DIFFRACTION
AUTHOR    D.R.TOMCHICK,R.J.TURNER,R.W.SWITZER,J.L.SMITH
...
SEQRES   1 A  181  MET ASN GLN LYS ALA VAL ILE LEU ASP GLU GLN ALA ILE
SEQRES   2 A  181  ARG ARG ALA LEU THR ARG ILE ALA HIS GLU MET ILE GLU
SEQRES   3 A  181  ARG ASN LYS GLY MET ASN ASN CYS ILE LEU VAL GLY ILE
SEQRES   4 A  181  LYS THR ARG GLY ILE TYR LEU ALA LYS ARG LEU ALA GLU
...
HELIX    1   1 GLU A   10  ARG A   27  1                                  18
HELIX    2   2 THR A   41  GLU A   58  1                                  18
HELIX    3   3 ARG A  112  VAL A  124  1                                  13
...
SHEET    1   A 5 SER A 129  VAL A 134  0
SHEET    2   A 5 LYS A 100  ASP A 105  1  N  VAL A 101   O  SER A 129
SHEET    3   A 5 ILE A  35  ILE A  39  1  N  ILE A  35   O  ILE A 102
SHEET    4   A 5 THR A  63  THR A  70  1  N  THR A  63   O  LEU A  36
SHEET    5   A 5 LEU A  86  ASP A  91 -1  N  ASP A  91   O  GLU A  66
SHEET    1   B 3 LYS A 161  GLN A 165  0
SHEET    2   B 3 LEU A 174  TYR A 178 -1  N  TYR A 178   O  LYS A 161
SHEET    3   B 3 LYS A   4  LEU A   8 -1  N  LEU A   8   O  VAL A 175
...
ATOM      1  N   GLN A   3     -11.430   2.565  20.717  1.00 35.35           N
ATOM      2  CA  GLN A   3     -10.573   3.777  20.613  1.00 34.92           C
```

```
ATOM      3  C   GLN A   3      -9.142   3.422  20.196  1.00 33.85           C
ATOM      4  O   GLN A   3      -8.485   2.572  20.802  1.00 34.11           O
ATOM      5  CB  GLN A   3     -10.545   4.517  21.949  1.00 35.97           C
ATOM      6  CG  GLN A   3      -9.831   5.851  21.918  1.00 37.00           C
ATOM      7  CD  GLN A   3      -9.655   6.442  23.301  1.00 37.73           C
ATOM      8  OE1 GLN A   3      -9.899   7.623  23.514  1.00 38.58           O
...
```

Listing 2.1: PDB file format example

An ordinary PDB entry [PDBc] consists of hundreds or even thousands of like the ones seen in Listing 2.1. The most relevant ones to our problem are:

- HEADER, TITLE and AUTHOR records

  These records give us information about the name of the protein, the researchers who defined it and the date when they did it.

- SEQRES records

  These records contain information about the chains that constitute the protein and also a listing of the chemical structures that are linearly linked to form it. In the example, we only have one (chain A) but normally they span several lines.

- HELIX records

  These records are designed to pinpoint the position of helices in the biomolecule. Every helix is named, numbered and classified accordingly to its type. The amino acids where the helix begins and end are also recorded.

- SHEET records

  These records can be used to determine the position of sheets in the protein. Each sheet is named and numbered. Each line corresponds to a strand of the sheet and the amino acids where they begin and end are records as well as their sense with respect to the previous strand.

- ATOM records

  These records establish the *xyz* coordinates for the residues and the chain to which they belong.

### 2.8.2 ARFF File

Attribute-Relation File Format (ARFF) is a file format which identifies an ASCII text file that determines a list of objects that have a set of common attributes [ARF]. They are divided into two different sections:

- Header

The header of the file holds the name of the relation, a list attributes and their respective types. An example of this can be found in Listing 2.2.

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

Listing 2.2: ARFF file Header example

- Data

The data part of the file simply lists the respective values of each attribute separated by commas, for every instance. An example can be found in Listing 2.3.

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

Listing 2.3: ARFF file Data example

Comments throughout this type of files start with '%'

# Chapter 3

# Proposed Solution

In this chapter we will describe in a more detailed manner the problem with which we are posed and its intricacies.

We will also provide an explanation of the solution model developed that was intended to provide trustworthy predictions of protein structures based on data mining methods that are faster to run than the experimentally based methods used in the molecular biology field.

A step by step characterization of the implementation and its workings will be given in an intelligible form in such a manner that it is as simple as possible to understand to someone who has little knowledge about the details of the matter.

## 3.1 Problem

The problem we have addressed in this work is a relatively small part of the protein folding process. The protein folding process may seem simple to someone that takes a distanced look at it although it is in essence quite difficult. The series of actions by which the amino acids of the linear sequence naturally fold is complex, inadequately understood, and definitely amounts to a series of small, local and interacting events.

The most direct and mechanic approach to the problem has its cornerstone set on atomic force fields that have been modelled and use concepts of mechanics in its most strict sense. This approach aims to find the folded protein's conformation based on it reaching a state of minimum free energy. Well, this poses as a rather difficult task because the enormous amount of possible conformations is still a huge computational obstacle and the powerful energetic forces that are involved in the folding process entail a very delicate balance between them. The mentioned forces are normally too challenging to model in a proper way and even when attempted the accumulation of small approximation errors may compromise the resulting model.

Another existent approach lies in the use of information contained in already determined protein conformations. These structures may act as spatial templates, provide informations about plausible folds as well as protein geometry and chemistry.

It is a very relevant and attractive take on the matter seeing as that proteins show recurrent organization patterns and are classified into structural families. It is estimated that there are only about 1000 to 10000 different types of the referred families. Using these, homology modelling methods employ a primary sequence chain similarity between the analysed sequence and the structure's inherent sequence. When the similarity is high this is the most successful prediction method known to date but in spite of this, this method is of limited applicability since it is very uncommon for sequences to have a similarity that is high enough when compared to those of which the structure is already determined.

As mentioned, the whole protein folding problem is very complex. In this thesis we have selected a Data Mining classification problem related with the alpha helices that can be useful to understand the folding problem. We have not tried to address with this the integral folding problem.

## 3.2 Proposed Solution

To tackle the problem posed, we carried out a comparative study on how different models based on data mining algorithms would fare in making predictions of the more complex protein structures with the input being their primary sequence of amino acids. This approach was intended to provide results in a relatively shorter amount of time than those mentioned in the previous section.

In order to do so, we had to find relevant information in the available repositories (which were brought up in Section 2.3). Because the information gathered from those platforms was so vast, a filtering process needed to be devised so that we could work only with relatively small examples of proteins to make sure that the work that was meant to be developed did not prove itself fruitless derived from both the complexity of the proteins and the relatively small processing power at our disposal.

After that filtering process was complete, the information needed to be parsed from the repositories and that was done recurring to programming code in Python and taking into account the singularities of each type of data, possible discrepancies and our needs in regard to the solution we wanted to construct.

Intertwined with the parsing process was the task of populating the database with the data that was deemed of interest to the field we are focused in, even if not all of that data would be used later for experimentation.

The next step in the process was to decide upon what experience we wanted to run. That meant we also had to establish what information from our database we would need, if it was enough and how we would do it.

Our experimentation part of the project took place with the WEKA environment. After studying the software and learn of its workings we made use of several of WEKA's solutions for data visualization, data preprocessing and evidently the data classification solution.

In the following section, further explanation of these steps are provided in a more detailed manner.

## 3.3 Implementation

### 3.3.1 Information gathering

The starting point of our information gathering was to choose one of the precompiled list of identified chains that were culled from the PDB archive by PISCES [WD03] which is a protein sequence culling server created a maintained by the Dunbrack Lab mentioned in Subsection 2.3.2.

There are quite a few lists available in their website, but since we wanted to use the data with less margin for error we picked the one that was more restrict in its parameters and thus had a smaller number of chains.

An entry of the list had the following parameters:

- Chain ID (that was in essence a concatenation of the protein ID and the chain identification code)
- Length (meaning the number of amino acids that constituted the chain)
- Experimental method
- Resolution
- R-factor (which is the measure of the quality of the atomic model obtained from the crystallographic data. The lower the better)
- Free R Value (which is similar to the R-factor but a less biased take)

All the chains recorded in the list were identified by X-ray diffraction, the percent identity cutoff is 20%, the resolution cutoff is 1.8 ångström and the R-factor cutoff is 0.25. Since there are weekly updates to the lists, it matters to say that the used list was released in May 2nd, 2018.

The idea of using this list was to get a collection of proteins and chains to use as a starting point when retrieving data from the PDB since it now has thousands of structures deposited into it has we've mentioned in Subsection 2.3.1.

### 3.3.2 PDB file parsing

Making use of Python's useful capabilities we had to parse from the precompiled file, download from the Dunbrack Lab website, the chain identifiers and store them in a data structure.

Using the identifiers stored we had to check for duplicates. After this, we cycled through these identifiers concatenating them with a string with which together formed the URL for that protein's PDB file download.

Now each of the PDB files corresponding to a protein contained a wide array of different type of records as seen in the example excerpt shown in Listing 2.1. To each of the lines of the file corresponded a different record and each type of record had its own specific construct. So, in order to parse the information we wanted from each line of the file, we required to implement a fitting solution. Later on in the dissertation, we will specify which of the information available in each kind of record we parsed.

- SEQRES

  For these type of entries which may represent represent a single chain or one of many parts of a chain, we parsed the following fields:

  - The identifier of that entry's chain
  - The 3 letter identifiers of the amino acids of that line were parsed into a data structure which could be updated in case that the next line's SEQRES entry was still referring to the same chain

- HELIX

  For these kind of records, a single entry was with certainty representing a single helix. So there was no necessity of checking if the following lines were still relevant. The fields we parsed were:

  - The helix number
  - The helix identifier
  - The 3 letter identifier of the amino acid where the helix started
  - The identifier of the protein chain where the helix started
  - The position in form of sequence number of the amino acid where the helix started (in relation to the beginning of the chain)
  - The 3 letter identifier of the amino acid where the helix ended
  - The identifier of the protein chain where the helix ended
  - The position in form of sequence number of the amino acid where the helix ended (in relation to the beginning of the chain)
  - The helix class that could be one of ten. These classes can be seen in Table 3.1.

Table 3.1: Type of helices

| Type of Helix | Class number |
|---|---|
| Right-handed alpha (default) | 1 |
| Right-handed omega | 2 |
| Right-handed pi | 3 |
| Right-handed gamma | 4 |
| Right-handed 3-10 | 5 |
| Left-handed alpha | 6 |
| Left-handed omega | 7 |
| Left-handed gamma | 8 |
| 2-7 ribbon/helix | 9 |
| Polyproline | 10 |

- SHEET

  In parsing these type of records we had to, once again, be aware if the next line would be referring to the same secondary structure, because each of these lines represents, in essence, only a strand of the sheet. So, the fields we parsed were:

  - The sheet identifier
  - The number of strands in the sheet
  - The 3 letter identifier of the amino acid where the strand started
  - The identifier of the protein chain where the strand started
  - The position in form of sequence number of the amino acid where the strand started (in relation to the beginning of the chain)
  - The 3 letter identifier of the amino acid where the strand ended
  - The identifier of the protein chain where the strand ended
  - The position in form of sequence number of the amino acid where the strand ended (in relation to the beginning of the chain)
  - The sense of the strand with respect to the previous one. The value of this was as shown in Table 3.2.

Table 3.2: Strand sense

| Sense of the strand | Corresponding number |
|---|---|
| If it was the first strand (default) | 0 |
| Parallel | 1 |
| Anti-parallel | -1 |

- ATOM

  For these type of records each line corresponds to a single atom so with regard to that it was not necessary to take into account what information would be in the next record. The attribute we parsed were:

  - The identifier of the protein chain to which the atom belonged
  - The atom's serial number
  - The chemical element to which it corresponded
  - The atom's name
  - The atom's coordinates for the $x$ axis in ångströms
  - The atom's coordinates for the $y$ axis in ångströms
  - The atom's coordinates for the $z$ axis in ångströms

### 3.3.3 Database

In order to store the data we had parsed, we created a database in MySQL. The tables were created taking into account the available data types in MySQL and how they would fit our needs.

The fact that we chose to store the parsed information in a local database was because going forward it would reduce immensely the access time when compared to accessing the information in the Web every time. Even in the smallest of the operations that change in performance was clear.

The tables' relations, columns and respective data types chosen for receiving the parsed data were as described in Figure 3.1.

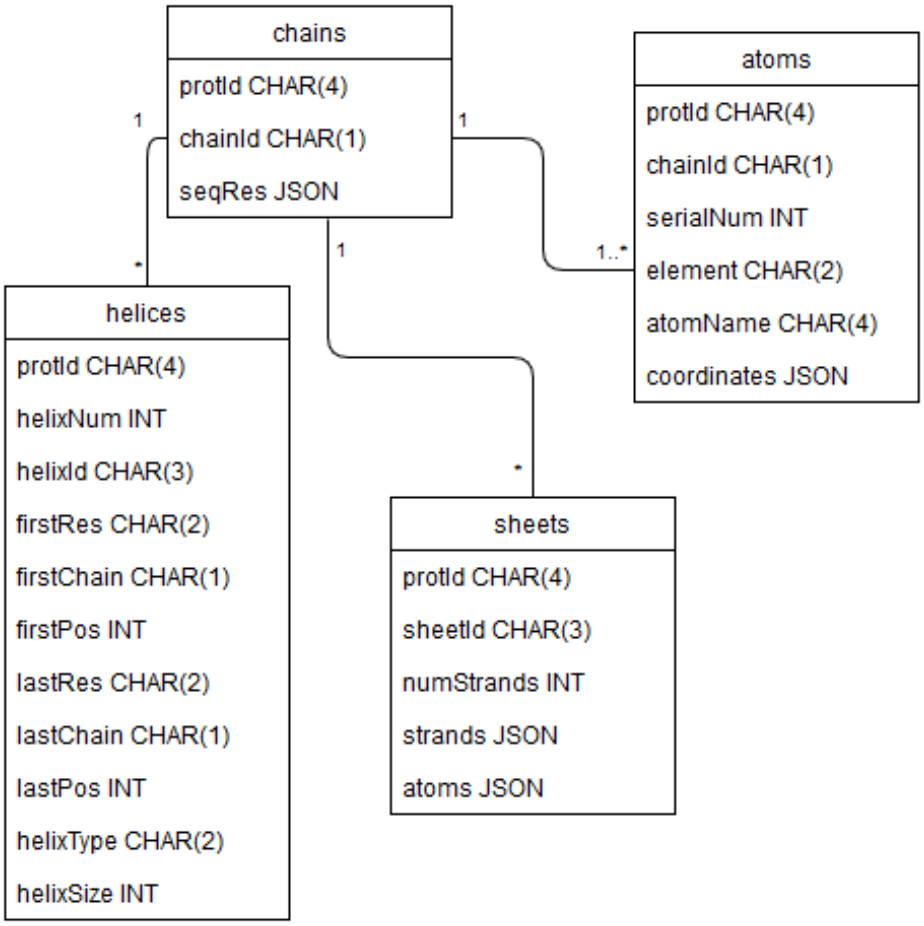


Figure 3.1: Database UML diagram

### 3.3.4 Datasets creation and preprocessing

With the data having already been stored in tables of the database, we had to construct the datasets to use on the experiences we wanted to conduct.

The experiments we wanted to run required us to relate data stored in the chains tables with data stored in the helices table. We wanted to assemble two datasets containing sequences of twenty consecutive amino acids. For the first one we wanted a sequence where the first ten appear just before the helix (meaning that they did not belong to an helix) and the following ten are the first ten amino acids of the helix (meaning that they did belong) and for the second one we wanted the opposite: the first ten amino acids should belong to an helix and the following ten should not.

In order to do so we needed to know exactly which amino acids belonged to helices and which did not. The way we did this was by comparing the positions where the helices started and ended against the corresponding amino acids chain of the protein.

In this process it was noticed that some of the entries of our helices table did not find the correct amino acids in the specified positions on the sequences stored in the chains table. Due to this we had to do this additional verification and discard the entries where this fault occurred.

The form by how this was done was by creating a data structure with the same size as the chain sequence where we stored a value that gave us that information. We either stored a 0 if the amino acid did not belong to any helix and a 1 for the cases where the amino acid did belong to one. Having done that we could now find the subsequences we wanted simply by comparing the original sequences against a pattern sequence.

In order to come up with datasets that were rich in information we decided to include in the dataset additional information for each of the residues that constituted the subsequences. The additional information for each amino acid is listed in Table 3.3. In the case of the acidity constants, what was used was instead the z-score for those values calculated with the help of the Python's library called NumPy[1].

The datasets construction was concluded by creating a nominal class attribute which evaluated to positive (POS) or negative (NEG). As there were far more subsequences that did not match the desired pattern than those that did, we made sure that we only had the same amount of instances classified as negative as the ones that were classified as positive. After the datasets were stored in our database, they were exported as a CSV file and then converted to ARFF files using a WEKA's tool with which it works. The dataset referring to the beginning of the helices contained 9004 instances of which half were classified as POS and the other half as NEG. As for the dataset that referred to the ending of the helices contained 8684 instances also with a 50/50 split regarding the classification of the instances as POS or NEG. The Python code through which this was done can be consulted in Appendix A.

[1] http://www.numpy.org/

Table 3.3: Additional physical properties of amino acids [Ami]

| Amino acid | Hydropathy | Charge | pKa, NH2 | pKa, COOH |
|---|---|---|---|---|
| Arginine | hydrophilic | positive | 9.09 | 2.18 |
| Asparagine | hydrophilic | neutral | 8.8 | 2.02 |
| Aspartate | hydrophilic | negative | 9.6 | 1.88 |
| Glutamate | hydrophilic | negative | 9.67 | 2.19 |
| Glutamine | hydrophilic | neutral | 9.13 | 2.17 |
| Lysine | hydrophilic | positive | 10.28 | 8.9 |
| Serine | hydrophilic | neutral | 9.15 | 2.21 |
| Threonine | hydrophilic | neutral | 9.12 | 2.15 |
| Cysteine | moderate | neutral | 10.78 | 1.71 |
| Histidine | moderate | positive | 8.97 | 1.78 |
| Methionine | moderate | neutral | 9.21 | 2.28 |
| Alanine | hydrophobic | neutral | 9.87 | 2.35 |
| Valine | hydrophobic | neutral | 9.72 | 2.29 |
| Glycine | hydrophobic | neutral | 9.6 | 2.34 |
| Isoleucine | hydrophobic | neutral | 9.76 | 2.32 |
| Leucine | hydrophobic | neutral | 9.6 | 2.36 |
| Phenylalanine | hydrophobic | neutral | 9.24 | 2.58 |
| Proline | hydrophobic | neutral | 10.6 | 1.99 |
| Tryptophan | hydrophobic | neutral | 9.39 | 2.38 |
| Tyrosine | hydrophobic | neutral | 9.11 | 2.2 |

#### 3.3.4.1 Attribute selection

In order to diversify our approach and in hopes of maybe getting better results, the original baseline datasets were put through a filtering process to remove some of the attributes. Using WEKA we loaded the ARFF files and in the separator "Select Attribute" we ran the *CorrelationAttributeEval* as the attribute evaluator and **Ranker** as the search method. We then ran the evaluator on the full datasets. The attributes selected for each dataset were the ones below. As we should, we maintained the attribute which represented the class of each instance which is not included in the lists below.

- **dataset_begin_helix**:
  - hydropathy11
  - hydropathy10
  - zScore_pKaNH2_11
  - hydropathy12
  - hydropathy14
  - hydropathy19
  - charge14
  - charge12
  - hydropathy17
  - charge15
  - charge19
  - zScore_pKaNH2_12
  - charge17
  - res11
  - hydropathy15
  - zScore_pKaCOOH_11
- **dataset_end_helix:**
  - zScore_pKaNH2_10
  - zScore_pKaNH2_8
  - hydropathy1
  - hydropathy10
  - hydropathy14
  - charge1
  - charge7

- hydropathy7
- charge14
- hydropathy5
- zScore_pKaCOOH_8
- res9
- zScore_pKaCOOH_6
- hydropathy11
- charge5
- res11
- charge8
- charge11

#### 3.3.4.2 Attribute enrichment

The last datasets were constructed by adding attributes to our files that relate the amino acids that constitute the subsequences stored. This was done in hopes of finding some sort of relation or pattern between amino acids that are close to each other so that we could maybe get improved results.

The way this attribute enrichment was done was by adding attributes which compared the informations of each amino acids to the ones of the following two. This means that for an amino acid *n* we created the following attributes:

- hydropathyCompare*n-n+1*, which compared the hydropathy of both amino acids. Evaluating to 1 if equal or to 0 if different.
- chargeCompare*n-n+1*, which compared the charge of both amino acids. Evaluating to 1 if equal or to 0 if different.
- zScoreNH2Compare*n-n+1*, which was the absolute value of the difference between the corresponding z-scores of the acidity constants for NH2 in both amino acids.
- zScoreCOOHCompare*n-n+1*, which was the absolute value of the difference between the corresponding z-scores of the acidity constants for COOH in both amino acids.
- hydropathyCompare*n-n+2*, which compared the hydropathy of both amino acids. Evaluating to 1 if equal or to 0 if different.
- chargeCompare*n-n+2*, which compared the charge of both amino acids. Evaluating to 1 if equal or to 0 if different.
- zScoreNH2Compare*n-n+2*, which was the absolute value of the difference between the corresponding z-scores of the acidity constants for NH2 in both amino acids.

- zScoreCOOHCompare*n*-*n*+*2*, which was the absolute value of the difference between the corresponding z-scores of the acidity constants for COOH in both amino acids. 2

# Chapter 4

# Experiments and Results

In this chapter we will present the experiments we ran with our datasets in order to evaluate the classifying models as well as the conditions on which we did it. We will further show the results obtained on the different experiments and the way we analysed them.

## 4.1 Testing Environment

All the experiments that will be presented were run in the same machine using the algorithms WEKA provides. The running of the algorithms was done by writing small programs in Java using WEKA's API which saved the results of each algorithm directly in text files. The machine on which the experiments were done had the specifications shown in Table 4.1.

Table 4.1: Testing machine specifications

| | |
|---|---|
| OS | Microsoft Windows 10 Pro |
| CPU | AMD FX 6300 Six-Core Processor, 3.50 GHz |
| Memory | 8 GB |

## 4.2 Testing specifications

So that we can understand how the tests were run on our datasets, we first need to understand how we treated the datasets that were created and already mentioned in Subsection 3.3.4. The way it was decided that the tests were going to be conducted was by applying the Monte Carlo method explained in Subsubsection 2.5.1.3. It was defined that, for each dataset, we would apply a randomizing filter to mix the instances five times, saving between each of those times the new randomized dataset before applying the filter again.

For every one of the referred five instances of randomized datasets we would then run a battery of classifying algorithms using a split percentage of 80/20. It matters to say that the two parts are mutually exclusive. The meaning the split holds is that eighty percent of the randomized dataset would be used as the training set and the remaining twenty percent of it would be used as the test set. The instances that belonged to those twenty percent would, therefore, be classified according to the model built based on the training made with the instances that belonged to the other eighty percent of the randomized dataset.

This sequence is represented by the diagram in Figure 4.1.

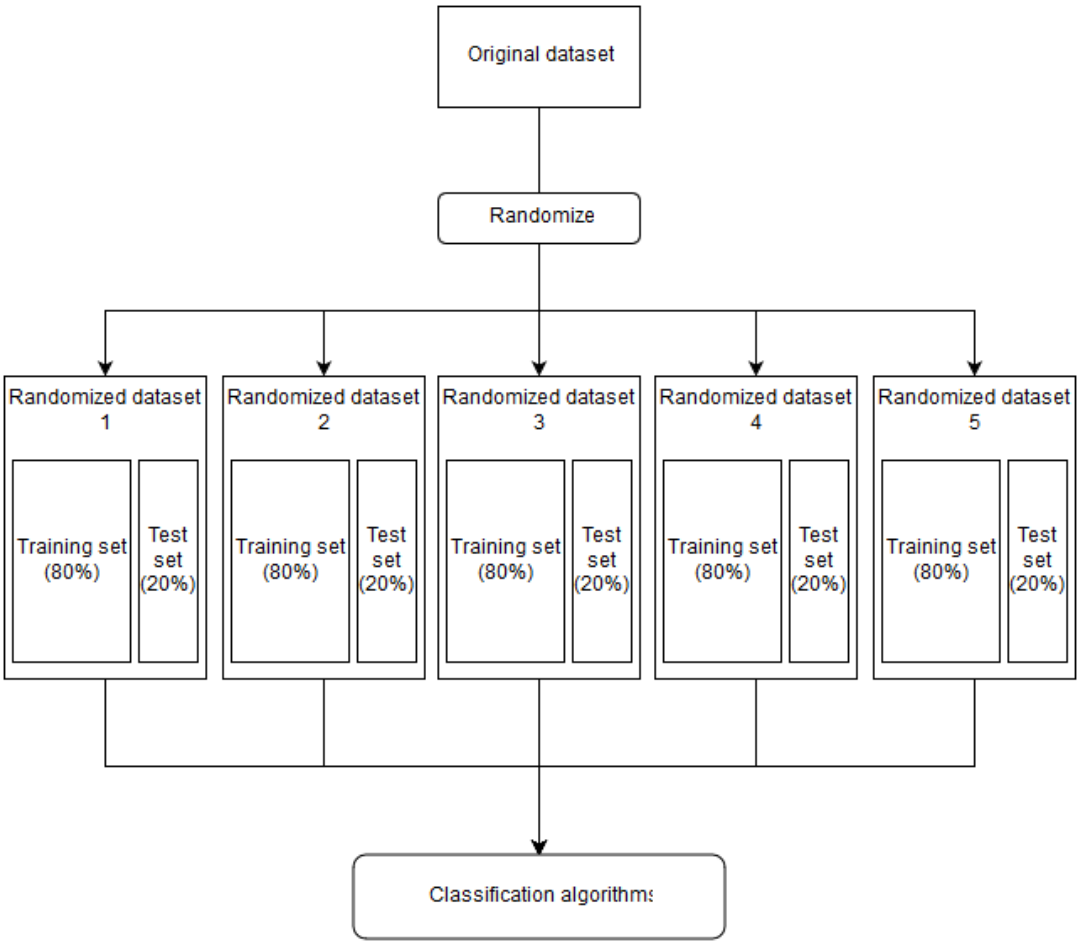


Figure 4.1: Diagram of the Monte Carlo method applied to the datasets

### 4.2.1 Classification algorithms

In order to try various approaches so that we could compare the resulting models fidelity, we first had to decide on which classification algorithms we would run on our datasets. We decided to run seven different classification algorithms from WEKA varying some of their parameters (as seen in Table 4.2) in expectation of maximizing their accuracy. The algorithms decided upon were:

- AdaBoost
- Bagging
- J48 (WEKA's implementation of C4.5)
- *k*-NN
- Multilayer Perceptron (Neural Network)
- Random Forest
- SVM

Table 4.2: Variation of parameters in the classification algorithms applied to the datasets

| **Classification algorithm** | **Parameters** | **Values** |
|---|---|---|
| **AdaBoost** | classifier | Decision Stump, J48 |
| | numIterations | 10, 100 |
| **Bagging** | numIterations | 10, 100 |
| **J48 (C4.5)** | minNumObj | 2, 10, 20 |
| ***k*-NN** | KNN | 1, 7 |
| | distanceFunction | Euclidean Distance, Manhattan Distance |
| **Multilayer Perceptron (Neural network)** | hiddenLayers | 10 |
| **Random Forest** | numIterations | 10, 500, 1000 |
| **SVM** | kernelType | PolyKernel, Normalized-PolyKernel |

## 4.3 Obtained results

The results of these tests ran on our multiple datasets were evaluated recurring to the comparison of metrics that WEKA outputs. The metrics we considered most relevant to compare were the Accuracy, the Precision, the F-measure and the AUROC.

The comparison was done by averaging the best result from the battery of algorithms run on each of the five randomized instances of every original dataset and also calculating the standard deviation (SD).

The results we obtained from the models created for each dataset are shown in Tables 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8. [2]

Table 4.3: Experiment 1.1 - Results from original dataset BeginHelix

| **Algorithm** | **Average ± Standard deviation** | | | |
|---|---|---|---|---|
| | **Accuracy** | **Precision** | **F-measure** | **AUROC** |
| **AdaBoost** | 0.865 ± 0.005 | 0.865 ± 0.005 | 0.865 ± 0.005 | 0.930 ± 0.005 |
| **Bagging** | 0.813 ± 0.004 | 0.813 ± 0.004 | 0.812 ± 0.004 | 0.895 ± 0.004 |
| **J48 (C4.5)** | 0.792 ± 0.006 | 0.792 ± 0.006 | 0.792 ± 0.006 | 0.814 ± 0.008 |
| ***k*-NN** | 0.807 ± 0.007 | 0.808 ± 0.007 | 0.807 ± 0.007 | 0.813 ± 0.007 |
| **Multilayer Perceptron (Neural Network)** | 0.750 ± 0.041 | 0.769 ± 0.009 | 0.746 ± 0.050 | 0.836 ± 0.005 |
| **Random Forest** | 0.870 ± 0.007 | 0.870 ± 0.007 | 0.870 ± 0.007 | 0.952 ± 0.003 |
| **SVM** | 0.822 ± 0.004 | 0.823 ± 0.004 | 0.822 ± 0.005 | 0.822 ± 0.004 |

Table 4.4: Experiment 1.2 - Results from original dataset EndHelix

| **Algorithm** | **Average ± Standard deviation** | | | |
|---|---|---|---|---|
| | **Accuracy** | **Precision** | **F-measure** | **AUROC** |
| **AdaBoost** | 0.797 ± 0.004 | 0.798 ± 0.003 | 0.797 ± 0.004 | 0.881 ± 0.003 |
| **Bagging** | 0.778 ± 0.004 | 0.778 ± 0.004 | 0.777 ± 0.004 | 0.864 ± 0.007 |
| **J48 (C4.5)** | 0.748 ± 0.011 | 0.749 ± 0.011 | 0.748 ± 0.011 | 0.777 ± 0.011 |
| ***k*-NN** | 0.752 ± 0.008 | 0.752 ± 0.007 | 0.752 ± 0.008 | 0.758 ± 0.008 |
| **Multilayer Perceptron (Neural Network)** | 0.701 ± 0.023 | 0.711 ± 0.024 | 0.697 ± 0.026 | 0.776 ± 0.022 |
| **Random Forest** | 0.808 ± 0.007 | 0.808 ± 0.007 | 0.808 ± 0.007 | 0.904 ± 0.006 |
| **SVM** | 0.763 ± 0.005 | 0.763 ± 0.005 | 0.763 ± 0.005 | 0.763 ± 0.005 |

Table 4.5: Experiment 2.1 - Results from dataset BeginHelix with attribute selection

| **Algorithm** | **Average ± Standard deviation** | | | |
|---|---|---|---|---|
| | **Accuracy** | **Precision** | **F-measure** | **AUROC** |
| **AdaBoost** | 0.827 ± 0.004 | 0.827 ± 0.004 | 0.827 ± 0.004 | 0.883 ± 0.002 |
| **Bagging** | 0.788 ± 0.006 | 0.789 ± 0.006 | 0.788 ± 0.006 | 0.867 ± 0.005 |
| **J48 (C4.5)** | 0.769 ± 0.008 | 0.770 ± 0.008 | 0.769 ± 0.008 | 0.805 ± 0.012 |
| ***k*-NN** | 0.813 ± 0.008 | 0.814 ± 0.008 | 0.813 ± 0.008 | 0.845 ± 0.009 |
| **Multilayer Perceptron (Neural Network)** | 0.748 ± 0.007 | 0.749 ± 0.006 | 0.747 ± 0.007 | 0.812 ± 0.004 |
| **Random Forest** | 0.836 ± 0.008 | 0.837 ± 0.008 | 0.836 ± 0.008 | 0.909 ± 0.006 |
| **SVM** | 0.732 ± 0.007 | 0.733 ± 0.007 | 0.733 ± 0.008 | 0.733 ± 0.008 |

Table 4.6: Experiment 2.2 - Results from dataset EndHelix with attribute selection

| **Algorithm** | **Average ± Standard deviation** | | | |
|---|---|---|---|---|
| | **Accuracy** | **Precision** | **F-measure** | **AUROC** |
| **AdaBoost** | 0.793 ± 0.006 | 0.793 ± 0.005 | 0.792 ± 0.006 | 0.865 ± 0.011 |
| **Bagging** | 0.728 ± 0.005 | 0.729 ± 0.005 | 0.728 ± 0.005 | 0.799 ± 0.006 |
| **J48 (C4.5)** | 0.747 ± 0.004 | 0.747 ± 0.004 | 0.747 ± 0.004 | 0.782 ± 0.004 |
| ***k*-NN** | 0.761 ± 0.005 | 0.761 ± 0.005 | 0.761 ± 0.005 | 0.783 ± 0.003 |
| **Multilayer Perceptron (Neural Network)** | 0.698 ± 0.014 | 0.700 ± 0.014 | 0.697 ± 0.015 | 0.757 ± 0.013 |
| **Random Forest** | 0.793 ± 0.006 | 0.793 ± 0.006 | 0.793 ± 0.006 | 0.890 ± 0.006 |
| **SVM** | 0.668 ± 0.007 | 0.669 ± 0.007 | 0.668 ± 0.007 | 0.668 ± 0.007 |

Table 4.7: Experiment 3.1 - Results from dataset BeginHelix with attribute enrichment

| **Algorithm** | **Average ± Standard deviation** | | | |
|---|---|---|---|---|
| | **Accuracy** | **Precision** | **F-measure** | **AUROC** |
| **AdaBoost** | 0.856 ± 0.004 | 0.856 ± 0.004 | 0.856 ± 0.004 | 0.923 ± 0.003 |
| **Bagging** | 0.815 ± 0.003 | 0.815 ± 0.003 | 0.815 ± 0.003 | 0.894 ± 0.003 |
| **J48 (C4.5)** | 0.801 ± 0.008 | 0.801 ± 0.008 | 0.801 ± 0.008 | 0.810 ± 0.005 |
| ***k*-NN** | 0.805 ± 0.007 | 0.806 ± 0.007 | 0.806 ± 0.007 | 0.811 ± 0.007 |
| **Multilayer Perceptron (Neural Network)** | 0.779 ± 0.027 | 0.784 ± 0.024 | 0.779 ± 0.027 | 0.840 ± 0.025 |
| **Random Forest** | 0.870 ± 0.005 | 0.870 ± 0.005 | 0.870 ± 0.005 | 0.951 ± 0.003 |
| **SVM** | 0.815 ± 0.005 | 0.816 ± 0.005 | 0.815 ± 0.005 | 0.815 ± 0.005 |

Table 4.8: Experiment 3.2 - Results from dataset EndHelix with attribute enrichment

| **Algorithm** | **Average ± Standard deviation** | | | |
|---|---|---|---|---|
| | **Accuracy** | **Precision** | **F-measure** | **AUROC** |
| **AdaBoost** | 0.786 ± 0.002 | 0.786 ± 0.003 | 0.786 ± 0.002 | 0.871 ± 0.003 |
| **Bagging** | 0.778 ± 0.005 | 0.779 ± 0.005 | 0.778 ± 0.005 | 0.863 ± 0.007 |
| **J48 (C4.5)** | 0.753 ± 0.006 | 0.753 ± 0.006 | 0.753 ± 0.006 | 0.768 ± 0.009 |
| ***k*-NN** | 0.753 ± 0.008 | 0.753 ± 0.008 | 0.753 ± 0.008 | 0.759 ± 0.009 |
| **Multilayer Perceptron (Neural Network)** | 0.732 ± 0.008 | 0.736 ± 0.009 | 0.731 ± 0.009 | 0.781 ± 0.009 |
| **Random Forest** | 0.808 ± 0.003 | 0.808 ± 0.004 | 0.808 ± 0.003 | 0.901 ± 0.003 |
| **SVM** | 0.762 ± 0.005 | 0.763 ± 0.004 | 0.762 ± 0.005 | 0.762 ± 0.005 |

## 4.4 Results Discussion

Right away, with a quick look at the results, we can tell that the algorithms which achieved the best results were the Random Forest and AdaBoost regardless of the experiment.

In Experiment 1.1, where the datasets used were the original without any type of preprocessing and corresponding to the BeginHelix case, we got the best results with Random Forest and AdaBoost with 0.870 and 0.865 respectively. These were very close, more so if we take into account the values of the standard deviation which were 0.007 and 0.005 respectively. As for the other evaluation metrics both algorithms continued to show very similar values except in the AUROC where the Random Forest gives a 0.952 opposing to the 0.930 from AdaBoost. From the remaining algorithms, Multilayer Perceptron was clearly the one with the worst results and SVM, Bagging and *k*-NN, though not as accurate as Random Forest and AdaBoost, still managed to provide some fairly good results, all above 0.800 accuracy.

For the Experiment 1.2, the algorithm that clearly stands out the most is the Random Forest with an accuracy of 0.808. AdaBoost is the second best with an accuracy of 0.797. Multilayer Perceptron is the worst with its accuracy values trailing by 0.047 points to the second worst which is the J48. The J48, *k*-NN and SVM results are kind of level while Bagging shows a somewhat significant better performance but still not enough to match the Bagging results.

The results of Experiment 2.1 which was the first with the datasets that went through a process of attribute selection, still show that the best one are coming from the Random Forest and AdaBoost, but with a lower accuracy when compared to the datasets without this type of preprocessing. Actually, every single one of the algorithms produced worse results with these datasets where the attributes were hand picked except for the *k*-NN which saw a small rise.

As for the Experiment 2.2 that ran the battery of algorithms of the datasets on the EndHelix datasets with attribute selection, Random Forest and AdaBoost are even more level as the ones that produced the best results. The only metrics that stands out to distinguish them is the AUROC where the Random Forest has a value of 0.890 and AdaBoost has 0.865. Like in Experiment 2.1, all algorithms but one - the *k*-NN - seem to produce less accurate classifications when compared to the the respective results of the datasets without preprocessing. The algorithm's results that appear to suffer the most with this change in the datasets are the results from the SVM, which sees its accuracy and the remaining evaluation metrics drop drastically.

In Experiment 3.1 the algorithms were run on the BeginHelix datasets with attribute enrichment. As expected, the general outcome was a better performance. Though the overall best result - Random Forest - had virtually the same values in every metric when compared against the original datasets without preprocessing, the greater part of the algorithms provided better results. The exceptions to this were the AdaBoost, SVM and *k*-NN that saw a dip in the values of the evaluation metrics but a very minimal one. Overall, the gains in the algorithms that performed better were much bigger than the losses in the ones that lowered their performance.

The conclusions drawn from the results of the Experiment 3.2 are practically the same as the ones drawn from the results of the Experiment 3.1. Random Forest is still the algorithm which

classifies an instance correctly the most times and the accuracy levels are on par with the ones for the original datasets without any preprocessing. The attribute enrichment ends up giving a boost in performance to the generality of the classifying algorithms, although not to the best overall. The Multilayer Perceptron algorithm sees a worthy of mention rise in accuracy with the attribute enrichment although it is still the algorithm which provides the lowest accuracy.

In a broad sense analysis, the results from the datasets BeginHelix are always higher than the results from the datasets EndHelix. Be that in the original conditions, with attribute selection or with attribute enrichment.

# Chapter 5

# Conclusions and Future Work

In this final chapter, it will be presented the conclusions of the dissertation, a comparison between the established objectives and the ones that were achieved as well as possible future work to be developed that can somehow give continuity to what was done.

## 5.1 Conclusions

Protein structure prediction is a big challenge for Molecular Biology. The determination of a protein's structure assumes a key role in many aspects of the research conducted nowadays in various of biology's areas of action. The fact that there are already a considerable amount of data available about proteins scattered across repositories gave to computer-based methods of prediction an important role.

The work we proposed to develop had the objective of contributing to the scientific area with yet another methodology for the prediction of protein structures as well as providing those predictions as accurately as possible in a time frame that would not be as slow as usual and costly in terms of processing power. The way we proposed to do it was by applying data mining classification algorithms to the data regarding proteins available in web repositories.

In order to do so we first had to parse those informations and store them in a database. Using the stored information, we had to create datasets that contained relevant information regarding the protein structure. In this case, we used the begging and ending of the helixes existent in the proteins.

The created datasets were put through a variety of filtering and preprocessing. The preprocessing included methods like attribute selection through the evaluation of the attributes or attribute enrichment in hopes of creating relations of which more information could be extracted.

Afterwards, we ran a battery of classification algorithms on the datasets varying the type of algorithm as well as some of theirs parameters.

The results achieved were very satisfactory, being able to build classification models that had an accuracy of up to 87%. The running of each of these classification algorithms took times that ranged between a few minutes to something like two hours. Therefore, the objective we proposed to achieve of providing a good prediction of protein structures in a timely fashion was achieved with success.

From the results we obtained we can also conclude that, for the datasets used, the algorithm of those tried that produces the best model is without a doubt the Random Forest. It consistently provided the results with the best accuracy across all datasets.

Regarding the preprocessing done in hopes of bettering the results when compared to the original datasets, the datasets with attribute selection provided worse results in general while the datasets with attribute enrichment provided better general results across the algorithms although it did not took the overall best result to higher values. This rise in performance should be related to the attributes introduced in the datasets that compared the attributes of each amino acid in the subsequence with the two that followed it.

## 5.2 Future Work

Though we feel that the objectives were met, further work and experimentation could be done in an attempt to produce even more accurate classification models and thus better structure prediction.

A starting point could be the creation of more datasets using the information stored in the database, which was not used in its entirety.

The preprocessing of the datasets could also be further developed, for example in the attribute enrichment it could be possible to try newer methods of comparison between the amino acids that belong to the sequence, using other extra information like we did with charge or the hydropathy or simply just by comparing each amino acid with more of the remaining amino acids in the sequence.

The classification algorithms are other aspect where more work and experimenting can be done, be that by trying new algorithms or fine tuning them by testing a variety their parameters to find which provides the most accurate results.

# References


[AJL+08] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell, 5th Edition*. 2008.

[Alt92] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *American Statistician*, 46(3):175–185, 1992.

[Ami] Amino acid physical properties. https://www.thermofisher.com/pt/en/home/life-science/protein-biology/protein-biology-learning-center/protein-biology-resource-library/pierce-protein-methods/amino-acid-physical-properties.html#table. Accessed: 2018-06-20.

[Anf73] Anfinsen. ' CIE : NCES Folding of Protein Chains. *Science*, 181(4096):223–230, 1973.

[ARF] Attribute-relation file format (arff). https://www.cs.waikato.ac.nz/ml/weka/arff.html. Accessed: 2018-06-20.

[AS08] Ana Azevedo and Manuel Filipe Santos. KDD, SEMMA and CRISP-DM: a parallel overview. *IADIS European Conference Data Mining*, (January):182–185, 2008.

[Ber08] Helen M. Berman. The Protein Data Bank: A historical perspective. *Acta Crystallographica Section A: Foundations of Crystallography*, 64(1):88–95, 2008.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[BvdSvD95] H. J.C. Berendsen, D. van der Spoel, and R. van Drunen. GROMACS: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 91(1-3):43–56, 1995.

[CCK+00] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. Crisp-Dm 1.0. *CRISP-DM Consortium*, page 76, 2000.

[CFR+12] Rui Camacho, Rita Ferreira, Natacha Rosa, Vânia Guimarães, Nuno a. Fonseca, Vítor Santos Costa, Miguel De Sousa, and Alexandre Magalhães. Predicting the secondary structure of proteins using machine learning algorithms. *International Journal of Data Mining and Bioinformatics*, 6(6):571–584, 2012.

[Dil07] The protein folding problem: when will it be solved? *Current Opinion in Structural Biology*, 17(3):342–346, 2007.

# REFERENCES


[dun] Dunbrack lab. http://dunbrack.fccc.edu/Home.php. Accessed: 2018-06-20.

[Dže09] Sašo Džeroski. Relational Data Mining. In *Data Mining and Knowledge Discovery Handbook*, pages 887–911. 2009.

[Edi05] Editorial. So Much More to Know. *Science*, 309(5731):78b–102b, 2005.

[FPSS96] Usama Fayyad, G Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, pages 37–54, 1996.

[Han07] David J. Hand. Principles of data mining. *Drug Safety*, 30(7):621–622, 2007.

[Ho98] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[Ho02] Tin Kam Ho. A data complexity analysis of comparative advantages of decision forest constructors. *Pattern Analysis and Applications*, 5(2):102–112, 2002.

[Lev69] Cyrus Levinthal. How to fold graciously. *Mössbauer Spectroscopy in Biological Systems Proceedings*, 24(41):22–24, 1969.

[PDBa] Methods for determining atomic structures. https://pdb101.rcsb.org/learn/guide-to-understanding-pdb-data/methods-for-determining-structure. Accessed: 2018-06-20.

[PDBb] Pdb statistics: Overall growth of released structures per year. https://www.rcsb.org/stats/growth/overall. Accessed: 2018-06-20.

[PDBc] Protein data bank contents guide: Atomic coordinate entry format description. http://www.wwpdb.org/documentation/file-format-content/format33/v3.3.html. Accessed: 2018-06-20.

[Qui92] J Ross Quinlan. *C4.5: Programs for Machine Learning*, volume 1. 1992.

[Ric81] Jane S. Richardson. The anatomy and taxonomy of protein structure. *Advances in Protein Chemistry*, 34(C):167–339, 1981.

[Sch15] Jürgen Schmidhuber. Deep Learning in neural networks: An overview, 2015.

[SM00] M S Smyth and J H J Martin. Review x Ray crystallography. *J Clin Pathol: Mol Pathol*, 53:8–14, 2000.

[SS99] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[WD03] Guoli Wang and Roland L. Dunbrack. PISCES: A protein sequence culling server. *Bioinformatics*, 19(12):1589–1591, 2003.

[WFE05] Ian H. Witten, Frank, and Eibe. *Practical machine learning tools and techniques*. 2005.

[Zha08] Yang Zhang. Progress and challenges in protein structure prediction. *Current Opinion in Structural Biology*, 18(3):342–348, 2008.

# Appendix A

# Datasets creation

Here will be presented the Python code used to create the original datasets without preprocessing. It stores the information in a table from the database, which then was exported in CSV file format a converted to ARRF file in WEKA.

## A.1 Dataset BeginHelix Constructor

```
import mysql.connector
import json
import time
import numpy as np
from scipy import stats

db = mysql.connector.connect(user='root', password='biotese123', host='localhost',
    database='proteins')

cursor = db.cursor(buffered=True)

cursorWrite = db.cursor(buffered=True)

cursor.execute("select chains.* from chains inner join helixes "
               "on chains.protId = helixes.protId and binary chains.chainId =
                   helixes.firstChain "
               "group by chains.protId, chains.chainId;")

chainsDict = {}
belongingDict = {}

commonResidues = ['ARG', 'ASN', 'ASP', 'GLU', 'GLN', 'LYS',
                  'SER', 'THR', 'CYS', 'HIS', 'MET', 'ALA',
                  'VAL', 'GLY', 'ILE', 'LEU', 'PHE', 'PRO',
                  'TRP', 'TYR']

```

```
hydropathy = ['hydrophilic', 'hydrophilic', 'hydrophilic', 'hydrophilic', '
    hydrophilic', 'hydrophilic',
              'hydrophilic', 'hydrophilic', 'moderate', 'moderate', 'moderate', '
                  hydrophobic',
              'hydrophobic', 'hydrophobic', 'hydrophobic', 'hydrophobic', '
                  hydrophobic', 'hydrophobic',
              'hydrophobic', 'hydrophobic']

charge = ['positive', 'neutral', 'negative', 'negative', 'neutral', 'positive',
          'neutral', 'neutral', 'neutral', 'positive', 'neutral', 'neutral',
          'neutral', 'neutral', 'neutral', 'neutral', 'neutral', 'neutral',
          'neutral', 'neutral']

pkaNH2 = [9.09, 8.8, 9.6, 9.67, 9.13, 10.28,
          9.15, 9.12, 10.78, 8.97, 9.21, 9.87,
          9.72, 9.6, 9.76, 9.6, 9.24, 10.6,
          9.39, 9.11]

pkaCOOH = [2.18, 2.02, 1.88, 2.19, 2.17, 8.9,
           2.21, 2.15, 1.71, 1.78, 2.28, 2.35,
           2.29, 2.34, 2.32, 2.36, 2.58, 1.99,
           2.38, 2.2]

mean = np.mean(pkaNH2)
std = np.std(pkaNH2)

zScore_pkaNH2 = stats.zscore(pkaNH2)
zScore_pkaCOOH = stats.zscore(pkaCOOH)

print(zScore_pkaNH2)

hydropathyDict = {commonResidues[i]: hydropathy[i] for i in range(20)}
chargeDict = {commonResidues[i]: charge[i] for i in range(20)}
zScore_pkaNH2_Dict = {commonResidues[i]: zScore_pkaNH2[i] for i in range(20)}
zScore_pkaCOOH_Dict = {commonResidues[i]: zScore_pkaCOOH[i] for i in range(20)}

print(hydropathyDict)
print(chargeDict)
print(zScore_pkaNH2_Dict)
print(zScore_pkaCOOH_Dict)


row = cursor.fetchone()

while row is not None:
    protId = row[0]
    chainId = row[1]
    seqRes: json = json.loads(row[2])
    belongingCodes = [0]*len(seqRes)
```

```
    key = (protId, chainId)
    chainsDict[key] = seqRes
    belongingDict[key] = belongingCodes

    row = cursor.fetchone()


cursor.execute("select distinct helixes.protId, chains.chainId, helixes.firstRes, "
               "helixes.firstPos, helixes.lastRes, helixes.lastPos, helixes.
                   helixSize from chains "
               "inner join helixes on chains.protId = helixes.protId "
               "and binary chains.chainId = helixes.firstChain")

row = cursor.fetchone()

while row is not None:
    protId = row[0]
    chainId = row[1]
    firstPos = row[3]
    lastPos = row[5]
    key = (protId, chainId)

    i = 0
    while i < len(belongingDict[key]):
        if (firstPos - 1) <= i <= (lastPos - 1):
            belongingDict[key][i] = 1
        i = i + 1

    row = cursor.fetchone()


cursor.execute("select chains.* from chains inner join helixes "
               "on chains.protId = helixes.protId and binary chains.chainId =
                   helixes.firstChain "
               "group by chains.protId, chains.chainId;")

row = cursor.fetchone()
pattern = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

rowCounter = 0
maxFile = 2067

positivesCounter = 0
negativesCounter = 0

while row is not None:
    protId = row[0]
    chainId = row[1]
```

```
    key = (protId, chainId)
    i = 20
    time.clock()
    rowCounter += 1

    print(str((rowCounter / maxFile) * 100)[:5] + '%')
    print('ETA:' + str((time.clock() / rowCounter) * (maxFile - rowCounter) / 60)
        [:6])

    while i < len(belongingDict[key]):
        if belongingDict[key][i-20:i] == pattern:
            res = chainsDict[key][i-20:i]
            sendToTable = True

            for x in res:
                if x not in commonResidues:
                    sendToTable = False
                    break

            if sendToTable:
                positivesCounter += 1

                entry = [res[0], hydropathyDict[res[0]], chargeDict[res[0]], float(
                    zScore_pkaNH2_Dict[res[0]]),
                         float(zScore_pkaCOOH_Dict[res[0]]),
                         res[1], hydropathyDict[res[1]], chargeDict[res[1]], float(
                             zScore_pkaNH2_Dict[res[1]]),
                         float(zScore_pkaCOOH_Dict[res[1]]),
                         res[2], hydropathyDict[res[2]], chargeDict[res[2]], float(
                             zScore_pkaNH2_Dict[res[2]]),
                         float(zScore_pkaCOOH_Dict[res[2]]),
                         res[3], hydropathyDict[res[3]], chargeDict[res[3]], float(
                             zScore_pkaNH2_Dict[res[3]]),
                         float(zScore_pkaCOOH_Dict[res[3]]),
                         res[4], hydropathyDict[res[4]], chargeDict[res[4]], float(
                             zScore_pkaNH2_Dict[res[4]]),
                         float(zScore_pkaCOOH_Dict[res[4]]),
                         res[5], hydropathyDict[res[5]], chargeDict[res[5]], float(
                             zScore_pkaNH2_Dict[res[5]]),
                         float(zScore_pkaCOOH_Dict[res[5]]),
                         res[6], hydropathyDict[res[6]], chargeDict[res[6]], float(
                             zScore_pkaNH2_Dict[res[6]]),
                         float(zScore_pkaCOOH_Dict[res[6]]),
                         res[7], hydropathyDict[res[7]], chargeDict[res[7]], float(
                             zScore_pkaNH2_Dict[res[7]]),
                         float(zScore_pkaCOOH_Dict[res[7]]),
                         res[8], hydropathyDict[res[8]], chargeDict[res[8]], float(
                             zScore_pkaNH2_Dict[res[8]]),
                         float(zScore_pkaCOOH_Dict[res[8]]),
```

```
                        res[9], hydropathyDict[res[9]], chargeDict[res[9]], float(
                            zScore_pkaNH2_Dict[res[9]]),
                        float(zScore_pkaCOOH_Dict[res[9]]),
                        res[10], hydropathyDict[res[10]], chargeDict[res[10]],
                            float(zScore_pkaNH2_Dict[res[10]]),
                        float(zScore_pkaCOOH_Dict[res[10]]),
                        res[11], hydropathyDict[res[11]], chargeDict[res[11]],
                            float(zScore_pkaNH2_Dict[res[11]]),
                        float(zScore_pkaCOOH_Dict[res[11]]),
                        res[12], hydropathyDict[res[12]], chargeDict[res[12]],
                            float(zScore_pkaNH2_Dict[res[12]]),
                        float(zScore_pkaCOOH_Dict[res[12]]),
                        res[13], hydropathyDict[res[13]], chargeDict[res[13]],
                            float(zScore_pkaNH2_Dict[res[13]]),
                        float(zScore_pkaCOOH_Dict[res[13]]),
                        res[14], hydropathyDict[res[14]], chargeDict[res[14]],
                            float(zScore_pkaNH2_Dict[res[14]]),
                        float(zScore_pkaCOOH_Dict[res[14]]),
                        res[15], hydropathyDict[res[15]], chargeDict[res[15]],
                            float(zScore_pkaNH2_Dict[res[15]]),
                        float(zScore_pkaCOOH_Dict[res[15]]),
                        res[16], hydropathyDict[res[16]], chargeDict[res[16]],
                            float(zScore_pkaNH2_Dict[res[16]]),
                        float(zScore_pkaCOOH_Dict[res[16]]),
                        res[17], hydropathyDict[res[17]], chargeDict[res[17]],
                            float(zScore_pkaNH2_Dict[res[17]]),
                        float(zScore_pkaCOOH_Dict[res[17]]),
                        res[18], hydropathyDict[res[18]], chargeDict[res[18]],
                            float(zScore_pkaNH2_Dict[res[18]]),
                        float(zScore_pkaCOOH_Dict[res[18]]),
                        res[19], hydropathyDict[res[19]], chargeDict[res[19]],
                            float(zScore_pkaNH2_Dict[res[19]]),
                        float(zScore_pkaCOOH_Dict[res[19]]),
                        'POS']

                print(entry)

                print(str((rowCounter / maxFile) * 100)[:5] + '%')
                print('ETA:' + str((time.clock() / rowCounter) * (maxFile -
                    rowCounter) / 60)[:6])

                cursorWrite.execute("insert into proteins.begin_helix ("
                                    "res1, hydropathy1, charge1, zScore_pKaNH2_1,
                                        zScore_pKaCOOH_1, "
                                    "res2, hydropathy2, charge2, zScore_pKaNH2_2,
                                        zScore_pKaCOOH_2, "
                                    "res3, hydropathy3, charge3, zScore_pKaNH2_3,
                                        zScore_pKaCOOH_3, "
```

```
                                "res4, hydropathy4, charge4, zScore_pKaNH2_4,
                                    zScore_pKaCOOH_4, "
                                "res5, hydropathy5, charge5, zScore_pKaNH2_5,
                                    zScore_pKaCOOH_5, "
                                "res6, hydropathy6, charge6, zScore_pKaNH2_6,
                                    zScore_pKaCOOH_6, "
                                "res7, hydropathy7, charge7, zScore_pKaNH2_7,
                                    zScore_pKaCOOH_7, "
                                "res8, hydropathy8, charge8, zScore_pKaNH2_8,
                                    zScore_pKaCOOH_8, "
                                "res9, hydropathy9, charge9, zScore_pKaNH2_9,
                                    zScore_pKaCOOH_9, "
                                "res10, hydropathy10, charge10,
                                    zScore_pKaNH2_10, zScore_pKaCOOH_10, "
                                "res11, hydropathy11, charge11,
                                    zScore_pKaNH2_11, zScore_pKaCOOH_11, "
                                "res12, hydropathy12, charge12,
                                    zScore_pKaNH2_12, zScore_pKaCOOH_12, "
                                "res13, hydropathy13, charge13,
                                    zScore_pKaNH2_13, zScore_pKaCOOH_13, "
                                "res14, hydropathy14, charge14,
                                    zScore_pKaNH2_14, zScore_pKaCOOH_14, "
                                "res15, hydropathy15, charge15,
                                    zScore_pKaNH2_15, zScore_pKaCOOH_15, "
                                "res16, hydropathy16, charge16,
                                    zScore_pKaNH2_16, zScore_pKaCOOH_16, "
                                "res17, hydropathy17, charge17,
                                    zScore_pKaNH2_17, zScore_pKaCOOH_17, "
                                "res18, hydropathy18, charge18,
                                    zScore_pKaNH2_18, zScore_pKaCOOH_18, "
                                "res19, hydropathy19, charge19,
                                    zScore_pKaNH2_19, zScore_pKaCOOH_19, "
                                "res20, hydropathy20, charge20,
                                    zScore_pKaNH2_20, zScore_pKaCOOH_20, "
                                "recordPositive) "
                                "values (%s, %s, %s, %s, %s, "
                                "%s, %s, %s, %s, %s, "
                                "%s, %s, %s, %s, %s, "
                                "%s, %s, %s, %s, %s, "
                                "%s, %s, %s, %s, %s, "
                                "%s, %s, %s, %s, %s, "
                                "%s, %s, %s, %s, %s, "
                                "%s, %s, %s, %s, %s, "
                                "%s, %s, %s, %s, %s, "
                                "%s, %s, %s, %s, %s, "
                                "%s, %s, %s, %s, %s, "
```

```
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s)", entry)

        elif belongingDict[key][i-20:i] != pattern:
            res = chainsDict[key][i - 20:i]
            sendToTable = True

            for x in res:
                if x not in commonResidues:
                    sendToTable = False
                    break

            if sendToTable and negativesCounter < positivesCounter:
                negativesCounter += 1

                entry = [res[0], hydropathyDict[res[0]], chargeDict[res[0]], float(
                    zScore_pkaNH2_Dict[res[0]]),
                         float(zScore_pkaCOOH_Dict[res[0]]),
                         res[1], hydropathyDict[res[1]], chargeDict[res[1]], float(
                             zScore_pkaNH2_Dict[res[1]]),
                         float(zScore_pkaCOOH_Dict[res[1]]),
                         res[2], hydropathyDict[res[2]], chargeDict[res[2]], float(
                             zScore_pkaNH2_Dict[res[2]]),
                         float(zScore_pkaCOOH_Dict[res[2]]),
                         res[3], hydropathyDict[res[3]], chargeDict[res[3]], float(
                             zScore_pkaNH2_Dict[res[3]]),
                         float(zScore_pkaCOOH_Dict[res[3]]),
                         res[4], hydropathyDict[res[4]], chargeDict[res[4]], float(
                             zScore_pkaNH2_Dict[res[4]]),
                         float(zScore_pkaCOOH_Dict[res[4]]),
                         res[5], hydropathyDict[res[5]], chargeDict[res[5]], float(
                             zScore_pkaNH2_Dict[res[5]]),
                         float(zScore_pkaCOOH_Dict[res[5]]),
                         res[6], hydropathyDict[res[6]], chargeDict[res[6]], float(
                             zScore_pkaNH2_Dict[res[6]]),
                         float(zScore_pkaCOOH_Dict[res[6]]),
                         res[7], hydropathyDict[res[7]], chargeDict[res[7]], float(
                             zScore_pkaNH2_Dict[res[7]]),
                         float(zScore_pkaCOOH_Dict[res[7]]),
                         res[8], hydropathyDict[res[8]], chargeDict[res[8]], float(
                             zScore_pkaNH2_Dict[res[8]]),
                         float(zScore_pkaCOOH_Dict[res[8]]),
                         res[9], hydropathyDict[res[9]], chargeDict[res[9]], float(
                             zScore_pkaNH2_Dict[res[9]]),
```

```
                        float(zScore_pkaCOOH_Dict[res[9]]),
                        res[10], hydropathyDict[res[10]], chargeDict[res[10]],
                            float(zScore_pkaNH2_Dict[res[10]]),
                        float(zScore_pkaCOOH_Dict[res[10]]),
                        res[11], hydropathyDict[res[11]], chargeDict[res[11]],
                            float(zScore_pkaNH2_Dict[res[11]]),
                        float(zScore_pkaCOOH_Dict[res[11]]),
                        res[12], hydropathyDict[res[12]], chargeDict[res[12]],
                            float(zScore_pkaNH2_Dict[res[12]]),
                        float(zScore_pkaCOOH_Dict[res[12]]),
                        res[13], hydropathyDict[res[13]], chargeDict[res[13]],
                            float(zScore_pkaNH2_Dict[res[13]]),
                        float(zScore_pkaCOOH_Dict[res[13]]),
                        res[14], hydropathyDict[res[14]], chargeDict[res[14]],
                            float(zScore_pkaNH2_Dict[res[14]]),
                        float(zScore_pkaCOOH_Dict[res[14]]),
                        res[15], hydropathyDict[res[15]], chargeDict[res[15]],
                            float(zScore_pkaNH2_Dict[res[15]]),
                        float(zScore_pkaCOOH_Dict[res[15]]),
                        res[16], hydropathyDict[res[16]], chargeDict[res[16]],
                            float(zScore_pkaNH2_Dict[res[16]]),
                        float(zScore_pkaCOOH_Dict[res[16]]),
                        res[17], hydropathyDict[res[17]], chargeDict[res[17]],
                            float(zScore_pkaNH2_Dict[res[17]]),
                        float(zScore_pkaCOOH_Dict[res[17]]),
                        res[18], hydropathyDict[res[18]], chargeDict[res[18]],
                            float(zScore_pkaNH2_Dict[res[18]]),
                        float(zScore_pkaCOOH_Dict[res[18]]),
                        res[19], hydropathyDict[res[19]], chargeDict[res[19]],
                            float(zScore_pkaNH2_Dict[res[19]]),
                        float(zScore_pkaCOOH_Dict[res[19]]),
                        'NEG']

                print(entry)

                print(str((rowCounter / maxFile) * 100)[:5] + '%')
                print('ETA:' + str((time.clock() / rowCounter) * (maxFile -
                    rowCounter) / 60)[:6])

                cursorWrite.execute("insert into proteins.begin_helix ("
                                    "res1, hydropathy1, charge1, zScore_pKaNH2_1,
                                        zScore_pKaCOOH_1, "
                                    "res2, hydropathy2, charge2, zScore_pKaNH2_2,
                                        zScore_pKaCOOH_2, "
                                    "res3, hydropathy3, charge3, zScore_pKaNH2_3,
                                        zScore_pKaCOOH_3, "
                                    "res4, hydropathy4, charge4, zScore_pKaNH2_4,
                                        zScore_pKaCOOH_4, "
```

```
294                                 "res5, hydropathy5, charge5, zScore_pKaNH2_5,
2                                       zScore_pKaCOOH_5, "
295                                 "res6, hydropathy6, charge6, zScore_pKaNH2_6,
4                                       zScore_pKaCOOH_6, "
296                                 "res7, hydropathy7, charge7, zScore_pKaNH2_7,
6                                       zScore_pKaCOOH_7, "
297                                 "res8, hydropathy8, charge8, zScore_pKaNH2_8,
8                                       zScore_pKaCOOH_8, "
298                                 "res9, hydropathy9, charge9, zScore_pKaNH2_9,
10                                      zScore_pKaCOOH_9, "
299                                 "res10, hydropathy10, charge10,
12                                      zScore_pKaNH2_10, zScore_pKaCOOH_10, "
300                                 "res11, hydropathy11, charge11,
14                                      zScore_pKaNH2_11, zScore_pKaCOOH_11, "
301                                 "res12, hydropathy12, charge12,
16                                      zScore_pKaNH2_12, zScore_pKaCOOH_12, "
302                                 "res13, hydropathy13, charge13,
18                                      zScore_pKaNH2_13, zScore_pKaCOOH_13, "
303                                 "res14, hydropathy14, charge14,
20                                      zScore_pKaNH2_14, zScore_pKaCOOH_14, "
304                                 "res15, hydropathy15, charge15,
22                                      zScore_pKaNH2_15, zScore_pKaCOOH_15, "
305                                 "res16, hydropathy16, charge16,
24                                      zScore_pKaNH2_16, zScore_pKaCOOH_16, "
306                                 "res17, hydropathy17, charge17,
26                                      zScore_pKaNH2_17, zScore_pKaCOOH_17, "
307                                 "res18, hydropathy18, charge18,
28                                      zScore_pKaNH2_18, zScore_pKaCOOH_18, "
308                                 "res19, hydropathy19, charge19,
30                                      zScore_pKaNH2_19, zScore_pKaCOOH_19, "
309                                 "res20, hydropathy20, charge20,
32                                      zScore_pKaNH2_20, zScore_pKaCOOH_20, "
310                                 "recordPositive) "
311                                 "values (%s, %s, %s, %s, %s, "
312                                 "%s, %s, %s, %s, %s, "
313                                 "%s, %s, %s, %s, %s, "
314                                 "%s, %s, %s, %s, %s, "
315                                 "%s, %s, %s, %s, %s, "
316                                 "%s, %s, %s, %s, %s, "
317                                 "%s, %s, %s, %s, %s, "
318                                 "%s, %s, %s, %s, %s, "
319                                 "%s, %s, %s, %s, %s, "
320                                 "%s, %s, %s, %s, %s, "
321                                 "%s, %s, %s, %s, %s, "
322                                 "%s, %s, %s, %s, %s, "
323                                 "%s, %s, %s, %s, %s, "
324                                 "%s, %s, %s, %s, %s, "
325                                 "%s, %s, %s, %s, %s, "
326                                 "%s, %s, %s, %s, %s, "
```

```
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s)", entry)
        i = i + 1
    row = cursor.fetchone()

db.commit()

db.close()
```

## A.2 Dataset EndHelix Constructor

```
import mysql.connector
import json
import time
import numpy as np
from scipy import stats

db = mysql.connector.connect(user='root', password='biotese123', host='localhost', database='proteins')

cursor = db.cursor(buffered=True)

cursorWrite = db.cursor(buffered=True)

cursor.execute("select chains.* from chains inner join helixes "
               "on chains.protId = helixes.protId and binary chains.chainId = helixes.firstChain "
               "group by chains.protId, chains.chainId;")

chainsDict = {}
belongingDict = {}

commonResidues = ['ARG', 'ASN', 'ASP', 'GLU', 'GLN', 'LYS',
                  'SER', 'THR', 'CYS', 'HIS', 'MET', 'ALA',
                  'VAL', 'GLY', 'ILE', 'LEU', 'PHE', 'PRO',
                  'TRP', 'TYR']

hydropathy = ['hydrophilic', 'hydrophilic', 'hydrophilic', 'hydrophilic', 'hydrophilic', 'hydrophilic',
              'hydrophilic', 'hydrophilic', 'moderate', 'moderate', 'moderate', 'hydrophobic',
              'hydrophobic', 'hydrophobic', 'hydrophobic', 'hydrophobic', 'hydrophobic', 'hydrophobic',
```

```
             'hydrophobic', 'hydrophobic']

charge = ['positive', 'neutral', 'negative', 'negative', 'neutral', 'positive',
          'neutral', 'neutral', 'neutral', 'positive', 'neutral', 'neutral',
          'neutral', 'neutral', 'neutral', 'neutral', 'neutral', 'neutral',
          'neutral', 'neutral']

pkaNH2 = [9.09, 8.8, 9.6, 9.67, 9.13, 10.28,
          9.15, 9.12, 10.78, 8.97, 9.21, 9.87,
          9.72, 9.6, 9.76, 9.6, 9.24, 10.6,
          9.39, 9.11]

pkaCOOH = [2.18, 2.02, 1.88, 2.19, 2.17, 8.9,
           2.21, 2.15, 1.71, 1.78, 2.28, 2.35,
           2.29, 2.34, 2.32, 2.36, 2.58, 1.99,
           2.38, 2.2]

mean = np.mean(pkaNH2)
std = np.std(pkaNH2)

zScore_pkaNH2 = stats.zscore(pkaNH2)
zScore_pkaCOOH = stats.zscore(pkaCOOH)

print(zScore_pkaNH2)

hydropathyDict = {commonResidues[i]: hydropathy[i] for i in range(20)}
chargeDict = {commonResidues[i]: charge[i] for i in range(20)}
zScore_pkaNH2_Dict = {commonResidues[i]: zScore_pkaNH2[i] for i in range(20)}
zScore_pkaCOOH_Dict = {commonResidues[i]: zScore_pkaCOOH[i] for i in range(20)}

print(hydropathyDict)
print(chargeDict)
print(zScore_pkaNH2_Dict)
print(zScore_pkaCOOH_Dict)


row = cursor.fetchone()

while row is not None:
    protId = row[0]
    chainId = row[1]
    seqRes: json = json.loads(row[2])
    belongingCodes = [0]*len(seqRes)
    key = (protId, chainId)
    chainsDict[key] = seqRes
    belongingDict[key] = belongingCodes

    row = cursor.fetchone()

```

```
77
78  cursor.execute("select distinct helixes.protId, chains.chainId, helixes.firstRes, "
79                 "helixes.firstPos, helixes.lastRes, helixes.lastPos, helixes.
                       helixSize from chains "
80                 "inner join helixes on chains.protId = helixes.protId "
81                 "and binary chains.chainId = helixes.firstChain")
82
83  row = cursor.fetchone()
84
85  while row is not None:
86      protId = row[0]
87      chainId = row[1]
88      firstPos = row[3]
89      lastPos = row[5]
90      key = (protId, chainId)
91
92      i = 0
93      while i < len(belongingDict[key]):
94          if (firstPos - 1) <= i <= (lastPos - 1):
95              belongingDict[key][i] = 1
96          i = i + 1
97
98      row = cursor.fetchone()
99
100
101 cursor.execute("select chains.* from chains inner join helixes "
102                "on chains.protId = helixes.protId and binary chains.chainId =
                       helixes.firstChain "
103                "group by chains.protId, chains.chainId;")
104
105 row = cursor.fetchone()
106 pattern = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
107            0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
108
109 rowCounter = 0
110 maxFile = 2067
111
112 positivesCounter = 0
113 negativesCounter = 0
114
115 while row is not None:
116     protId = row[0]
117     chainId = row[1]
118     key = (protId, chainId)
119     i = 20
120     time.clock()
121     rowCounter += 1
122
123     print(str((rowCounter / maxFile) * 100)[:5] + '%')
```

```
    print('ETA:' + str((time.clock() / rowCounter) * (maxFile - rowCounter) / 60)[:6])

    while i < len(belongingDict[key]):
        if belongingDict[key][i-20:i] == pattern:
            res = chainsDict[key][i-20:i]
            sendToTable = True

            for x in res:
                if x not in commonResidues:
                    sendToTable = False
                    break

            if sendToTable:
                positivesCounter += 1

                entry = [res[0], hydropathyDict[res[0]], chargeDict[res[0]], float(zScore_pkaNH2_Dict[res[0]]),
                         float(zScore_pkaCOOH_Dict[res[0]]),
                         res[1], hydropathyDict[res[1]], chargeDict[res[1]], float(zScore_pkaNH2_Dict[res[1]]),
                         float(zScore_pkaCOOH_Dict[res[1]]),
                         res[2], hydropathyDict[res[2]], chargeDict[res[2]], float(zScore_pkaNH2_Dict[res[2]]),
                         float(zScore_pkaCOOH_Dict[res[2]]),
                         res[3], hydropathyDict[res[3]], chargeDict[res[3]], float(zScore_pkaNH2_Dict[res[3]]),
                         float(zScore_pkaCOOH_Dict[res[3]]),
                         res[4], hydropathyDict[res[4]], chargeDict[res[4]], float(zScore_pkaNH2_Dict[res[4]]),
                         float(zScore_pkaCOOH_Dict[res[4]]),
                         res[5], hydropathyDict[res[5]], chargeDict[res[5]], float(zScore_pkaNH2_Dict[res[5]]),
                         float(zScore_pkaCOOH_Dict[res[5]]),
                         res[6], hydropathyDict[res[6]], chargeDict[res[6]], float(zScore_pkaNH2_Dict[res[6]]),
                         float(zScore_pkaCOOH_Dict[res[6]]),
                         res[7], hydropathyDict[res[7]], chargeDict[res[7]], float(zScore_pkaNH2_Dict[res[7]]),
                         float(zScore_pkaCOOH_Dict[res[7]]),
                         res[8], hydropathyDict[res[8]], chargeDict[res[8]], float(zScore_pkaNH2_Dict[res[8]]),
                         float(zScore_pkaCOOH_Dict[res[8]]),
                         res[9], hydropathyDict[res[9]], chargeDict[res[9]], float(zScore_pkaNH2_Dict[res[9]]),
                         float(zScore_pkaCOOH_Dict[res[9]]),
                         res[10], hydropathyDict[res[10]], chargeDict[res[10]],
                             float(zScore_pkaNH2_Dict[res[10]]),
                         float(zScore_pkaCOOH_Dict[res[10]]),
```

```
                        res[11], hydropathyDict[res[11]], chargeDict[res[11]],
                            float(zScore_pkaNH2_Dict[res[11]]),
                        float(zScore_pkaCOOH_Dict[res[11]]),
                        res[12], hydropathyDict[res[12]], chargeDict[res[12]],
                            float(zScore_pkaNH2_Dict[res[12]]),
                        float(zScore_pkaCOOH_Dict[res[12]]),
                        res[13], hydropathyDict[res[13]], chargeDict[res[13]],
                            float(zScore_pkaNH2_Dict[res[13]]),
                        float(zScore_pkaCOOH_Dict[res[13]]),
                        res[14], hydropathyDict[res[14]], chargeDict[res[14]],
                            float(zScore_pkaNH2_Dict[res[14]]),
                        float(zScore_pkaCOOH_Dict[res[14]]),
                        res[15], hydropathyDict[res[15]], chargeDict[res[15]],
                            float(zScore_pkaNH2_Dict[res[15]]),
                        float(zScore_pkaCOOH_Dict[res[15]]),
                        res[16], hydropathyDict[res[16]], chargeDict[res[16]],
                            float(zScore_pkaNH2_Dict[res[16]]),
                        float(zScore_pkaCOOH_Dict[res[16]]),
                        res[17], hydropathyDict[res[17]], chargeDict[res[17]],
                            float(zScore_pkaNH2_Dict[res[17]]),
                        float(zScore_pkaCOOH_Dict[res[17]]),
                        res[18], hydropathyDict[res[18]], chargeDict[res[18]],
                            float(zScore_pkaNH2_Dict[res[18]]),
                        float(zScore_pkaCOOH_Dict[res[18]]),
                        res[19], hydropathyDict[res[19]], chargeDict[res[19]],
                            float(zScore_pkaNH2_Dict[res[19]]),
                        float(zScore_pkaCOOH_Dict[res[19]]),
                        'POS']

                print(entry)

                print(str((rowCounter / maxFile) * 100)[:5] + '%')
                print('ETA:' + str((time.clock() / rowCounter) * (maxFile -
                    rowCounter) / 60)[:6])

                cursorWrite.execute("insert into proteins.end_helix ("
                                    "res1, hydropathy1, charge1, zScore_pkaNH2_1,
                                        zScore_pkaCOOH_1, "
                                    "res2, hydropathy2, charge2, zScore_pkaNH2_2,
                                        zScore_pkaCOOH_2, "
                                    "res3, hydropathy3, charge3, zScore_pkaNH2_3,
                                        zScore_pkaCOOH_3, "
                                    "res4, hydropathy4, charge4, zScore_pkaNH2_4,
                                        zScore_pkaCOOH_4, "
                                    "res5, hydropathy5, charge5, zScore_pkaNH2_5,
                                        zScore_pkaCOOH_5, "
                                    "res6, hydropathy6, charge6, zScore_pkaNH2_6,
                                        zScore_pkaCOOH_6, "
```

```
193                                 "res7, hydropathy7, charge7, zScore_pKaNH2_7,
  2                                     zScore_pKaCOOH_7, "
194                                 "res8, hydropathy8, charge8, zScore_pKaNH2_8,
  4                                     zScore_pKaCOOH_8, "
195                                 "res9, hydropathy9, charge9, zScore_pKaNH2_9,
  6                                     zScore_pKaCOOH_9, "
196                                 "res10, hydropathy10, charge10,
  8                                     zScore_pKaNH2_10, zScore_pKaCOOH_10, "
197                                 "res11, hydropathy11, charge11,
 10                                     zScore_pKaNH2_11, zScore_pKaCOOH_11, "
198                                 "res12, hydropathy12, charge12,
 12                                     zScore_pKaNH2_12, zScore_pKaCOOH_12, "
199                                 "res13, hydropathy13, charge13,
 14                                     zScore_pKaNH2_13, zScore_pKaCOOH_13, "
200                                 "res14, hydropathy14, charge14,
 16                                     zScore_pKaNH2_14, zScore_pKaCOOH_14, "
201                                 "res15, hydropathy15, charge15,
 18                                     zScore_pKaNH2_15, zScore_pKaCOOH_15, "
202                                 "res16, hydropathy16, charge16,
 20                                     zScore_pKaNH2_16, zScore_pKaCOOH_16, "
203                                 "res17, hydropathy17, charge17,
 22                                     zScore_pKaNH2_17, zScore_pKaCOOH_17, "
204                                 "res18, hydropathy18, charge18,
 24                                     zScore_pKaNH2_18, zScore_pKaCOOH_18, "
205                                 "res19, hydropathy19, charge19,
 26                                     zScore_pKaNH2_19, zScore_pKaCOOH_19, "
206                                 "res20, hydropathy20, charge20,
 28                                     zScore_pKaNH2_20, zScore_pKaCOOH_20, "
207                                 "recordPositive) "
208                                 "values (%s, %s, %s, %s, %s, "
209                                 "%s, %s, %s, %s, %s, "
210                                 "%s, %s, %s, %s, %s, "
211                                 "%s, %s, %s, %s, %s, "
212                                 "%s, %s, %s, %s, %s, "
213                                 "%s, %s, %s, %s, %s, "
214                                 "%s, %s, %s, %s, %s, "
215                                 "%s, %s, %s, %s, %s, "
216                                 "%s, %s, %s, %s, %s, "
217                                 "%s, %s, %s, %s, %s, "
218                                 "%s, %s, %s, %s, %s, "
219                                 "%s, %s, %s, %s, %s, "
220                                 "%s, %s, %s, %s, %s, "
221                                 "%s, %s, %s, %s, %s, "
222                                 "%s, %s, %s, %s, %s, "
223                                 "%s, %s, %s, %s, %s, "
224                                 "%s, %s, %s, %s, %s, "
225                                 "%s, %s, %s, %s, %s, "
226                                 "%s, %s, %s, %s, %s, "
227                                 "%s, %s, %s, %s, %s, "
```

```
                                    "%s)", entry)

        elif belongingDict[key][i-20:i] != pattern:
            res = chainsDict[key][i - 20:i]
            sendToTable = True

            for x in res:
                if x not in commonResidues:
                    sendToTable = False
                    break

            if sendToTable and negativesCounter < positivesCounter:
                negativesCounter += 1

                entry = [res[0], hydropathyDict[res[0]], chargeDict[res[0]], float(
                    zScore_pkaNH2_Dict[res[0]]),
                         float(zScore_pkaCOOH_Dict[res[0]]),
                         res[1], hydropathyDict[res[1]], chargeDict[res[1]], float(
                             zScore_pkaNH2_Dict[res[1]]),
                         float(zScore_pkaCOOH_Dict[res[1]]),
                         res[2], hydropathyDict[res[2]], chargeDict[res[2]], float(
                             zScore_pkaNH2_Dict[res[2]]),
                         float(zScore_pkaCOOH_Dict[res[2]]),
                         res[3], hydropathyDict[res[3]], chargeDict[res[3]], float(
                             zScore_pkaNH2_Dict[res[3]]),
                         float(zScore_pkaCOOH_Dict[res[3]]),
                         res[4], hydropathyDict[res[4]], chargeDict[res[4]], float(
                             zScore_pkaNH2_Dict[res[4]]),
                         float(zScore_pkaCOOH_Dict[res[4]]),
                         res[5], hydropathyDict[res[5]], chargeDict[res[5]], float(
                             zScore_pkaNH2_Dict[res[5]]),
                         float(zScore_pkaCOOH_Dict[res[5]]),
                         res[6], hydropathyDict[res[6]], chargeDict[res[6]], float(
                             zScore_pkaNH2_Dict[res[6]]),
                         float(zScore_pkaCOOH_Dict[res[6]]),
                         res[7], hydropathyDict[res[7]], chargeDict[res[7]], float(
                             zScore_pkaNH2_Dict[res[7]]),
                         float(zScore_pkaCOOH_Dict[res[7]]),
                         res[8], hydropathyDict[res[8]], chargeDict[res[8]], float(
                             zScore_pkaNH2_Dict[res[8]]),
                         float(zScore_pkaCOOH_Dict[res[8]]),
                         res[9], hydropathyDict[res[9]], chargeDict[res[9]], float(
                             zScore_pkaNH2_Dict[res[9]]),
                         float(zScore_pkaCOOH_Dict[res[9]]),
                         res[10], hydropathyDict[res[10]], chargeDict[res[10]],
                             float(zScore_pkaNH2_Dict[res[10]]),
                         float(zScore_pkaCOOH_Dict[res[10]]),
                         res[11], hydropathyDict[res[11]], chargeDict[res[11]],
                             float(zScore_pkaNH2_Dict[res[11]]),
```

```
                        float(zScore_pkaCOOH_Dict[res[11]]),
                        res[12], hydropathyDict[res[12]], chargeDict[res[12]],
                            float(zScore_pkaNH2_Dict[res[12]]),
                        float(zScore_pkaCOOH_Dict[res[12]]),
                        res[13], hydropathyDict[res[13]], chargeDict[res[13]],
                            float(zScore_pkaNH2_Dict[res[13]]),
                        float(zScore_pkaCOOH_Dict[res[13]]),
                        res[14], hydropathyDict[res[14]], chargeDict[res[14]],
                            float(zScore_pkaNH2_Dict[res[14]]),
                        float(zScore_pkaCOOH_Dict[res[14]]),
                        res[15], hydropathyDict[res[15]], chargeDict[res[15]],
                            float(zScore_pkaNH2_Dict[res[15]]),
                        float(zScore_pkaCOOH_Dict[res[15]]),
                        res[16], hydropathyDict[res[16]], chargeDict[res[16]],
                            float(zScore_pkaNH2_Dict[res[16]]),
                        float(zScore_pkaCOOH_Dict[res[16]]),
                        res[17], hydropathyDict[res[17]], chargeDict[res[17]],
                            float(zScore_pkaNH2_Dict[res[17]]),
                        float(zScore_pkaCOOH_Dict[res[17]]),
                        res[18], hydropathyDict[res[18]], chargeDict[res[18]],
                            float(zScore_pkaNH2_Dict[res[18]]),
                        float(zScore_pkaCOOH_Dict[res[18]]),
                        res[19], hydropathyDict[res[19]], chargeDict[res[19]],
                            float(zScore_pkaNH2_Dict[res[19]]),
                        float(zScore_pkaCOOH_Dict[res[19]]),
                        'NEG']

                print(entry)

                print(str((rowCounter / maxFile) * 100)[:5] + '%')
                print('ETA:' + str((time.clock() / rowCounter) * (maxFile -
                    rowCounter) / 60)[:6])

                cursorWrite.execute("insert into proteins.end_helix ("
                                    "res1, hydropathy1, charge1, zScore_pKaNH2_1,
                                        zScore_pKaCOOH_1, "
                                    "res2, hydropathy2, charge2, zScore_pKaNH2_2,
                                        zScore_pKaCOOH_2, "
                                    "res3, hydropathy3, charge3, zScore_pKaNH2_3,
                                        zScore_pKaCOOH_3, "
                                    "res4, hydropathy4, charge4, zScore_pKaNH2_4,
                                        zScore_pKaCOOH_4, "
                                    "res5, hydropathy5, charge5, zScore_pKaNH2_5,
                                        zScore_pKaCOOH_5, "
                                    "res6, hydropathy6, charge6, zScore_pKaNH2_6,
                                        zScore_pKaCOOH_6, "
                                    "res7, hydropathy7, charge7, zScore_pKaNH2_7,
                                        zScore_pKaCOOH_7, "
```

```
                                    "res8, hydropathy8, charge8, zScore_pKaNH2_8,
                                        zScore_pKaCOOH_8, "
                                    "res9, hydropathy9, charge9, zScore_pKaNH2_9,
                                        zScore_pKaCOOH_9, "
                                    "res10, hydropathy10, charge10,
                                        zScore_pKaNH2_10, zScore_pKaCOOH_10, "
                                    "res11, hydropathy11, charge11,
                                        zScore_pKaNH2_11, zScore_pKaCOOH_11, "
                                    "res12, hydropathy12, charge12,
                                        zScore_pKaNH2_12, zScore_pKaCOOH_12, "
                                    "res13, hydropathy13, charge13,
                                        zScore_pKaNH2_13, zScore_pKaCOOH_13, "
                                    "res14, hydropathy14, charge14,
                                        zScore_pKaNH2_14, zScore_pKaCOOH_14, "
                                    "res15, hydropathy15, charge15,
                                        zScore_pKaNH2_15, zScore_pKaCOOH_15, "
                                    "res16, hydropathy16, charge16,
                                        zScore_pKaNH2_16, zScore_pKaCOOH_16, "
                                    "res17, hydropathy17, charge17,
                                        zScore_pKaNH2_17, zScore_pKaCOOH_17, "
                                    "res18, hydropathy18, charge18,
                                        zScore_pKaNH2_18, zScore_pKaCOOH_18, "
                                    "res19, hydropathy19, charge19,
                                        zScore_pKaNH2_19, zScore_pKaCOOH_19, "
                                    "res20, hydropathy20, charge20,
                                        zScore_pKaNH2_20, zScore_pKaCOOH_20, "
                                    "recordPositive) "
                                    "values (%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s, %s, %s, %s, %s, "
                                    "%s)", entry)
        i = i + 1
```

```
    row = cursor.fetchone()

db.commit()

db.close()
```